pyanno4rt

Tim Ortkamp

May 14, 2024

CONTENTS

CHAPTER

ONE

GENERAL INFORMATION

pyanno4rt is a Python package for conventional and outcome prediction model-based inverse photon and proton treatment plan optimization, including radiobiological and machine learning (ML) models for tumor control probability (TCP) and normal tissue complication probability (NTCP). It leverages state-of-the-art local and global solution methods to handle both single- and multi-objective (un)constrained optimization problems, thereby covering a number of different problem designs. To summarize roughly, the following functionality is provided:

CHAPTER

TWO

INSTALLATION

You can install the latest distribution via:

```
pip install pyanno4rt
```

You can check the latest source code via:

```
git clone https://github.com/pyanno4rt/pyanno4rt.git
```

pyanno4rt has two main classes which provide a code-based and a UI-based interface:

Base class import for CLI/IDE

```
from pyanno4rt.base import TreatmentPlan
```

GUI import

```
from pyanno4rt.gui import GraphicalUserInterface
```

CHAPTER

THREE

DEVELOPMENT

CHAPTER

FOUR

CONTRIBUTING

pyanno4rt is open for new contributors of all experience levels. Please get in contact with us (see "Help and support") to discuss the format of your contribution.

Note: the "docs" folder on Github includes example files with CT/segmentation data and the photon dose-influence matrix for the TG-119 case, a standard test phantom which can be used for development. You will find more realistic patient data e.g. in the CORT dataset1 or the TROTS dataset2.

CHAPTER

FIVE

HELP AND SUPPORT

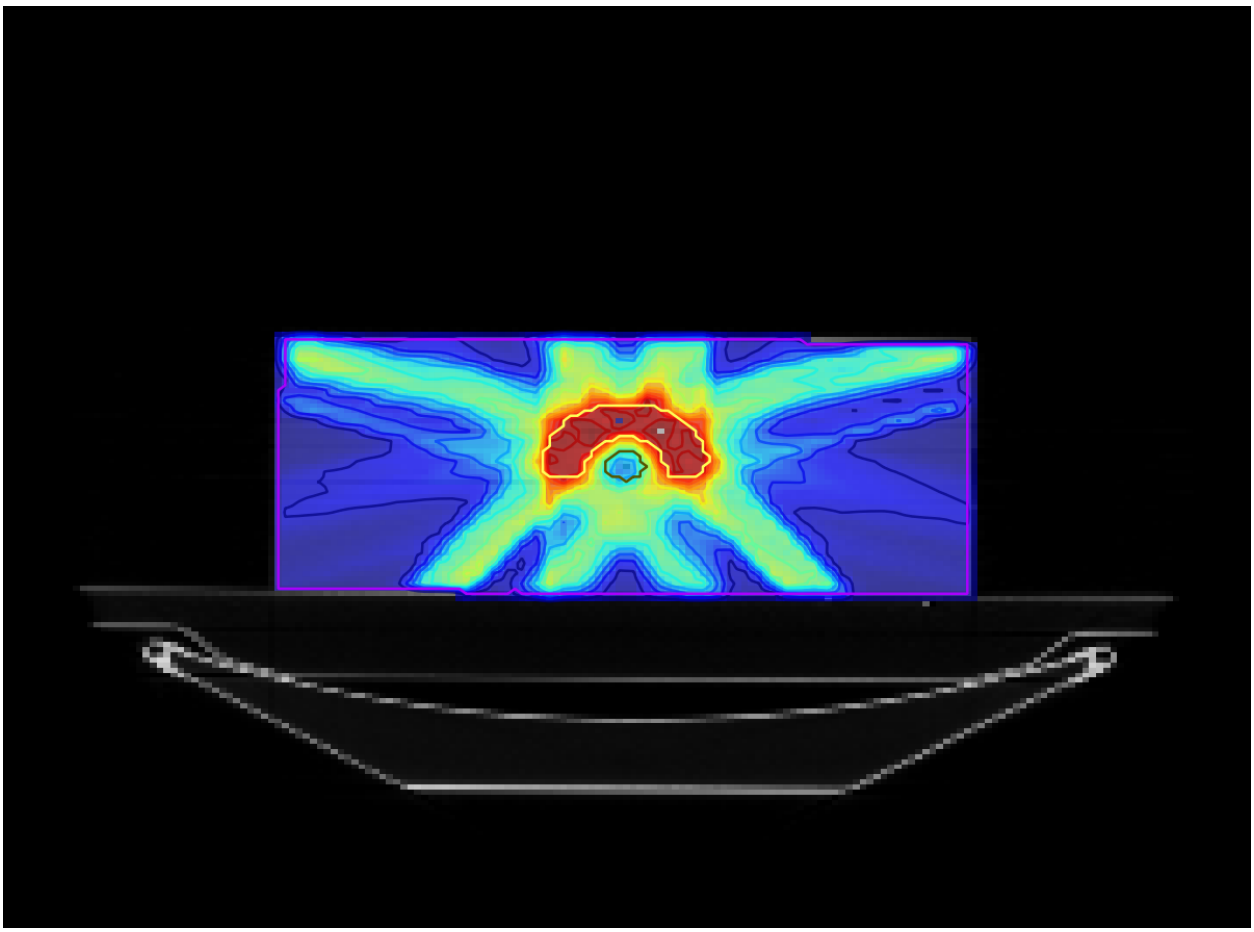
To cite this repository:

```bibtex
@misc{pyanno4rt2024,
  title = {{pyanno4rt}: python-based advanced numerical nonlinear optimization for radiotherapy},
  author = {Ortkamp, Tim and Jäkel, Oliver and Frank, Martin and Wahl, Niklas},
  year = {2024},
  howpublished = {\url{http://github.com/pyanno4rt/pyanno4rt}}
}
```

CHAPTER

SIX

EXAMPLE: TG-119 STANDARD TREATMENT PLAN OPTIMIZATION



6.1 Intro

Welcome to the pyanno4rt example notebook! In this notebook, we will showcase the core functionality of our package using data from the TG-119 standard case (available from our Github repository as .mat-files). The first part will present a beginner-friendly version of the code-based interface, followed by the UI-based interface in the second part.

6.2 Import of the relevant classes

First, we import the base classes. Our package is designed for clarity and ease of use, wherefore it has only one class for initializing a treatment plan and one class for initializing the graphical user interface. Hence, the import statements are:

```
from pyanno4rt.base import TreatmentPlan
from pyanno4rt.gui import GraphicalUserInterface
```

6.3 Code-based interface

If you prefer to work with the command line interface (CLI) or an interactive development environment (IDE), you can initialize the TreatmentPlan class by hand. The parameter space of this class is divided into three parameter groups:

Well then, let's create an instance of the TreatmentPlan class!

6.3.1 Treatment plan initialization

For the sake of readability, we will define the parameter groups one by one (of course, you could also directly specify them in the base class arguments). Our package utilizes Python dictionaries for this purpose, which allow an efficient mapping between parameter names and values per group and promote a transparent setup and passing.

Setting up the configuration dictionary

We decide to label our plan 'TG-119-example' and set the minimum logging level to 'info', which means that any debugging messages will be suppressed. For the modality and the number of fractions, we stick to the default values 'photon' and 30. Since we have some MATLAB files available for the TG-119 case, we provide the corresponding paths to the imaging and dose-influence matrix files (you may adapt them). Post-processing interpolation of the imaging data is not required, so we leave the parameter at None. Finally, we know that the dose-influence matrix has been calculated with a resolution of 6 mm in each dimension, so we set the dose resolution parameter accordingly.

```
configuration = {
    'label': 'TG-119-example', # Unique identifier for the treatment plan
    'min_log_level': 'info', # Minimum logging level
    'modality': 'photon', # Treatment modality
    'number_of_fractions': 30, # Number of fractions
    'imaging_path': './TG_119_data.mat', # Path to the CT and segmentation data
    'target_imaging_resolution': None, # Imaging resolution for post-processing␣
↪interpolation of the CT and segmentation data
    'dose_matrix_path': './TG_119_photonDij.mat', # Path to the dose-influence matrix
    'dose_resolution': [6, 6, 6] # Size of the dose grid in [mm] per dimension
    }
```

Great, we have completely defined the first parameter group

Setting up the optimization dictionary

Next, we need to describe how the TG-119 treatment plan should be optimized. In general, the final plan should apply a reasonably high dose to the target volumes while limiting the dose exposure to relevant organs at risk to prevent post-treatment complications. To achieve this, we define objective functions for the core ('Core'), for the outer target ('OuterTarget'), and for the whole body ('BODY'), where 'Squared Overdosing' refers to a function that penalizes dose values above a maximum, and 'Squared Deviation' refers to a function that penalizes upward and downward deviations from a target. The definition of these functions is again based on a dictionary, the components dictionary: it takes the segment names from the imaging data as keys and sub-dictionaries as values, in which the component type ('objective' or 'constraint') and the component instance (or a list of instances) with the component's class name and parameters are set. Once the components have been defined, we find ourselves in a trade-off situation, where a higher degree of fulfillment for one objective is usually accompanied with a lower degree of fulfillment for another. We can handle this by choosing the 'weighted-sum' method, which bypasses the multi-objective problem by multiplying each objective value with a weight parameter and then summing them up, effectively merging them into a scalar "total" objective function. This works well with the default solution algorithm, the 'L-BFGS-B' algorithm from the 'scipy' solver, so we pick that one. For the initialization of the fluence vector (holding the decision variables), we opt for 'target-coverage' to start off with a satisfactory dose level for the outer target (alternatively we could have passed 'warm-start' and replaced None for the initial fluence vector with an array). We place a lower bound of 0 and no upper bound (None) on the fluence, matching its physical properties. As the final step, we limit the number of iterations to 500 and the tolerance (precision goal) for the objective function value to 0.001.

```
optimization = {
    'components': { # Optimization components for each segment of interest
        'Core': {
            'type': 'objective',
            'instance': {
                'class': 'Squared Overdosing',
                'parameters': {
                    'maximum_dose': 25,
                    'weight': 100
                }
            }
        },
        'OuterTarget': {
            'type': 'objective',
            'instance': {
                'class': 'Squared Deviation',
                'parameters': {
                    'target_dose': 60,
                    'weight': 1000
                }
            }
        },
        'BODY': {
            'type': 'objective',
            'instance': {
                'class': 'Squared Overdosing',
                'parameters': {
                    'maximum_dose': 30,
                    'weight': 800
                }
            }
        }
```

(continues on next page)

(continued from previous page)

```
    },
    'method': 'weighted-sum', # Single- or multi-criteria optimization method
    'solver': 'scipy', # Python package to be used for solving the optimization problem
    'algorithm': 'L-BFGS-B', # Solution algorithm from the chosen solver
    'initial_strategy': 'target-coverage', # Initialization strategy for the fluence␣
↪vector
    'initial_fluence_vector': None, # User-defined initial fluence vector (only for 'warm-
↪start')
    'lower_variable_bounds': 0, # Lower bounds on the decision variables
    'upper_variable_bounds': None, # Upper bounds on the decision variables
    'max_iter': 500, # Maximum number of iterations for the solvers to converge
    'tolerance': 0.001 # Precision goal for the objective function value
    }
```

Yeah, this was a tough piece of work! If you have managed to complete the optimization dictionary, feel free to reward yourself with a cup of tea or coffee, maybe a small snack, and a relaxing short break before moving on

Setting up the evaluation dictionary

It is not actually necessary to set up the evaluation dictionary if you are happy with the default values. However, we will initialize it for reasons of completeness. First, we select the DVH type 'cumulative' and request its evaluation at 1000 (evenly-spaced) points. With the parameters 'reference_volume' and 'reference_dose', we let the package calculate dose and volume quantiles at certain levels. By inserting an empty list for 'reference_dose', the levels are automatically determined. The last two parameters, 'display_segments' and 'display_metrics', can be used to filter the names of the segments and metrics to be displayed later in the treatment plan visualization. We also specify empty lists here to not exclude any segment or metric.

```
evaluation = {
    'dvh_type': 'cumulative', # Type of DVH to be calculated
    'number_of_points': 1000, # Number of (evenly-spaced) points for which to evaluate␣
↪the DVH
    'reference_volume': [2, 5, 50, 95, 98], # Reference volumes for which to calculate␣
↪the inverse DVH values
    'reference_dose': [], # Reference dose values for which to calculate the DVH values
    'display_segments': [], # Names of the segmented structures to be displayed
    'display_metrics': [] # Names of the plan evaluation metrics to be displayed
    }
```

Congratulations, you have successfully set up all parameter dictionaries

Initializing the base class

Now let's finally put everything together into a complete TreatmentPlan instance.

```
tp = TreatmentPlan(configuration, optimization, evaluation)
```

6.3.2 Treatment plan workflow

In this section, we describe the standard workflow in which the generated treatment plan instance comes into play. Our package equips the instance with one method for each work step, which can be called parameter-free.

Configuring the plan

First, a successfully initialized treatment plan needs to be configured. By calling the configure method, the information from the configuration dictionary is transferred to internal instances of the configuration classes, which perform functional (logging, data management) and I/O tasks (processing of imaging data, preparation of data dictionaries). Note that a plan must be configured before it can be optimized.

```
tp.configure()
```

```
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Initializing logger ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - You are running Python version␣
↪3.10.14 ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Initializing datahub ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Initializing patient loader ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Initializing plan generator ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Initializing dose information␣
↪generator ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Importing CT and segmentation␣
↪data from MATLAB file ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Generating plan configuration␣
↪for photon treatment ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Generating dose information␣
↪for photon treatment ...
2024-05-14 20:59:29 - pyanno4rt - TG-119-example - INFO - Adding dose-influence matrix␣
↪from MATLAB file ...
```

Optimizing the plan

Afterwards, the treatment plan is ready for optimization. We call the optimize method, which generates the internal optimization classes, passes the optimization parameters from the dictionary, and at the end triggers the solver run. If machine learning model-based components are used, the model fitting would also take place here. In our example, no such components exist, which means that the optimization process starts immediately. Note that a plan must be optimized before it can be evaluated.

```
tp.optimize()
```

```
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Initializing fluence optimizer␣
↪...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Removing segment overlaps ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Resizing segments from CT to␣
↪dose grid ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Setting the optimization␣
↪components ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Setting objective 'Squared␣
↪Overdosing' for ['Core'] ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Setting objective 'Squared␣
```

(continues on next page)

(continued from previous page)

```
↪Deviation' for ['OuterTarget'] ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Setting objective 'Squared␣
↪Overdosing' for ['BODY'] ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Adjusting dose parameters for␣
↪fractionation ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Initializing dose projection ..
↪.
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Initializing weighted-sum␣
↪optimization method ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Initializing fluence␣
↪initializer ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Initializing fluence vector␣
↪with respect to target coverage ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Initializing SciPy solver with␣
↪L-BFGS-B algorithm ...
2024-05-14 20:59:31 - pyanno4rt - TG-119-example - INFO - Solving optimization problem ..
↪.
2024-05-14 20:59:32 - pyanno4rt - TG-119-example - INFO - At iterate 0: f=144.1014
2024-05-14 20:59:33 - pyanno4rt - TG-119-example - INFO - At iterate 1: f=119.7357
2024-05-14 20:59:33 - pyanno4rt - TG-119-example - INFO - At iterate 2: f=85.2347
2024-05-14 20:59:33 - pyanno4rt - TG-119-example - INFO - At iterate 3: f=30.4996
2024-05-14 20:59:33 - pyanno4rt - TG-119-example - INFO - At iterate 4: f=14.5176
2024-05-14 20:59:33 - pyanno4rt - TG-119-example - INFO - At iterate 5: f=12.4683
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 6: f=10.9733
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 7: f=10.3196
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 8: f=9.684
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 9: f=9.4383
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 10: f=8.9254
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 11: f=8.3518
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 12: f=8.0128
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 13: f=7.5543
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 14: f=7.3486
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 15: f=6.9583
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 16: f=6.5906
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 17: f=6.3163
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 18: f=6.1272
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 19: f=6.0453
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 20: f=5.9811
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 21: f=5.8058
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 22: f=5.7397
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 23: f=5.6837
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 24: f=5.6219
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 25: f=5.5861
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 26: f=5.5432
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 27: f=5.5025
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 28: f=5.4791
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 29: f=5.4677
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 30: f=5.4307
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 31: f=5.4234
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 32: f=5.4037
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 33: f=5.3914
2024-05-14 20:59:34 - pyanno4rt - TG-119-example - INFO - At iterate 34: f=5.3791
```

(continues on next page)

(continued from previous page)

```
2024-05-14 20:59:35 - pyanno4rt - TG-119-example - INFO - At iterate 35: f=5.3647
2024-05-14 20:59:35 - pyanno4rt - TG-119-example - INFO - At iterate 36: f=5.3593
2024-05-14 20:59:35 - pyanno4rt - TG-119-example - INFO - At iterate 37: f=5.3487
2024-05-14 20:59:35 - pyanno4rt - TG-119-example - INFO - At iterate 38: f=5.3436
2024-05-14 20:59:35 - pyanno4rt - TG-119-example - INFO - Computing 3D dose cube from␣
↪optimized fluence vector ...
2024-05-14 20:59:35 - pyanno4rt - TG-119-example - INFO - Fluence optimizer took 3.62␣
↪seconds (3.26 seconds for problem solving) ...
```

Evaluating the plan

The penultimate step usually is the evaluation of the treatment plan, and following the previous logic, we have added an evaluate method for this purpose. Internally, this creates objects from the DVH and dosimetrics class, which take the parameters of the evaluation dictionary and trigger the respective evaluation processes.

```
tp.evaluate()
```

```
2024-05-14 20:59:37 - pyanno4rt - TG-119-example - INFO - Initializing DVH evaluator ...
2024-05-14 20:59:37 - pyanno4rt - TG-119-example - INFO - Initializing dosimetrics␣
↪evaluator ...
2024-05-14 20:59:37 - pyanno4rt - TG-119-example - INFO - Evaluating cumulative DVH with␣
↪1000 points for all segments ...
2024-05-14 20:59:37 - pyanno4rt - TG-119-example - INFO - Evaluating dosimetrics for all␣
↪segments ...
```

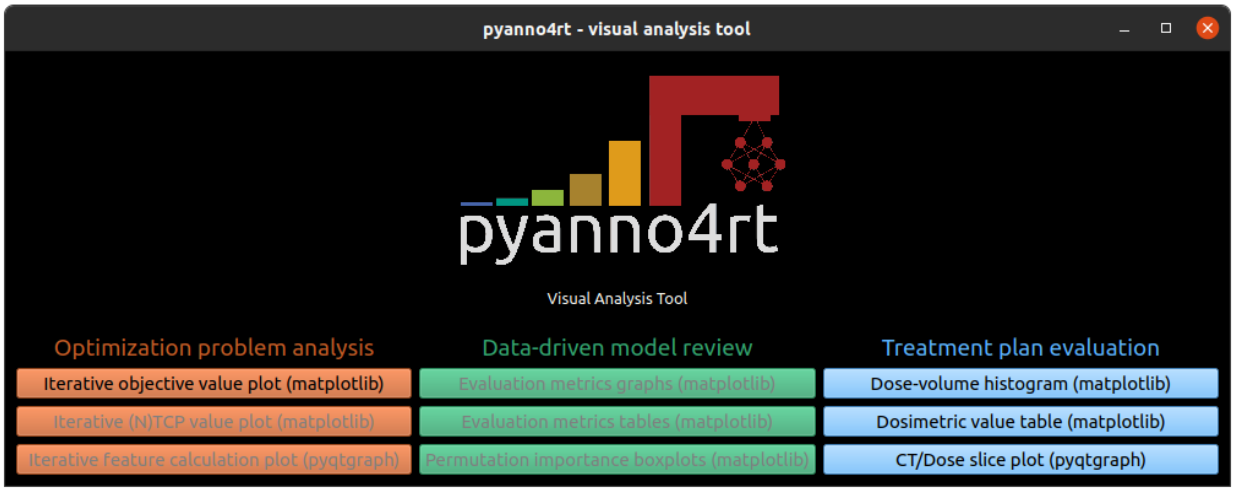
Visualizing the plan

We are now at the end of the standard workflow, and of course we would like to conclude by analyzing the results of the treatment plan optimization and evaluation both qualitatively and quantitatively. Our package features a visual analysis tool that provides three sets of visualizations: optimization problem analysis, data-driven model review, and treatment plan evaluation. By clicking on the activated buttons, you can open the plot windows. The visual analysis tool can easily be launched with the visualize method.

```
tp.visualize()
```

```
2024-05-14 20:59:37 - pyanno4rt - TG-119-example - INFO - Initializing visualizer ...
QPixmap::scaled: Pixmap is a null pixmap
2024-05-14 20:59:38 - pyanno4rt - TG-119-example - INFO - Launching visualizer ...
2024-05-14 20:59:39 - pyanno4rt - TG-119-example - INFO - Opening CT/dose slice plot ...
QPixmap::scaled: Pixmap is a null pixmap
2024-05-14 21:00:11 - pyanno4rt - TG-119-example - INFO - Closing visualizer ...
```

Ideally, you should now see the window below.



(By the way, the top image in this notebook has been extracted from the CT/Dose slice plot)

Shortcut: composing the plan

Many times you will just run all four of the above methods in sequence. To make this a little more convenient, the treatment plan can also be "composed" in a single step, using the appropriately named compose method (and yeah, we love music).

```
tp.compose()
```

Updating parameter values

One last class functionality is the updating of parameter values with the update method. This comes in handy because each of the configure, optimize and evaluate methods is based on the corresponding parameter dictionary, so that, for example, the evaluate method can be called again after updating an evaluation parameter without repeating all the initialization and prior workflow steps. The update method takes a dictionary with key-value pairs as input, where the former are from the parameter dictionaries, and the latter are the new parameter values. We do not want to change the plan at this point, so we will just overwrite the modality and the DVH type with the previous values for illustration purposes.

```
tp.update({
    'modality': 'photon',
    'dvh_type': 'cumulative'
    })
```

Saving and loading treatment plans

Treatment plans generated within our package can be saved as a snapshot folder and loaded from there as a copycat. You can import the corresponding functions from the tools subpackage.

```
from pyanno4rt.tools import copycat, snapshot
```

A snapshot automatically includes a JSON file with the parameter dictionaries, a compiled log file, and, if machine learning model-based components are used, subfolders with model configuration files. Optionally, you can specify whether to add the imaging data, the dose-influence matrix, and the model training data (this allows sharing an instance

of TreatmentPlan with all input data). The name of the snapshot folder is specified by the treatment plan label from the configuration dictionary. Assuming the snapshot is to be saved in the current path, the line below would create the minimum-sized version of a snapshot folder.

```
snapshot(instance=tp, path='./', include_patient_data=False, include_dose_matrix=False,␣
↪include_model_data=False)
```

Conversely, a snapshot that has been saved can be loaded back into a Python variable by calling the copycat function with the base class and the folder path.

```
tp_copy = copycat(base_class=TreatmentPlan, path='./TG-119-example/')
```

6.4 UI-based interface

Our package can also be accessed from a graphical user interface (GUI) if you prefer this option. There are many good reasons for using the GUI:

And, of course, a GUI may also simply look good

6.4.1 GUI initialization

So, how can the GUI be called? Instead of initializing the TreatmentPlan class, we create an object of the GraphicalUserInterface class.

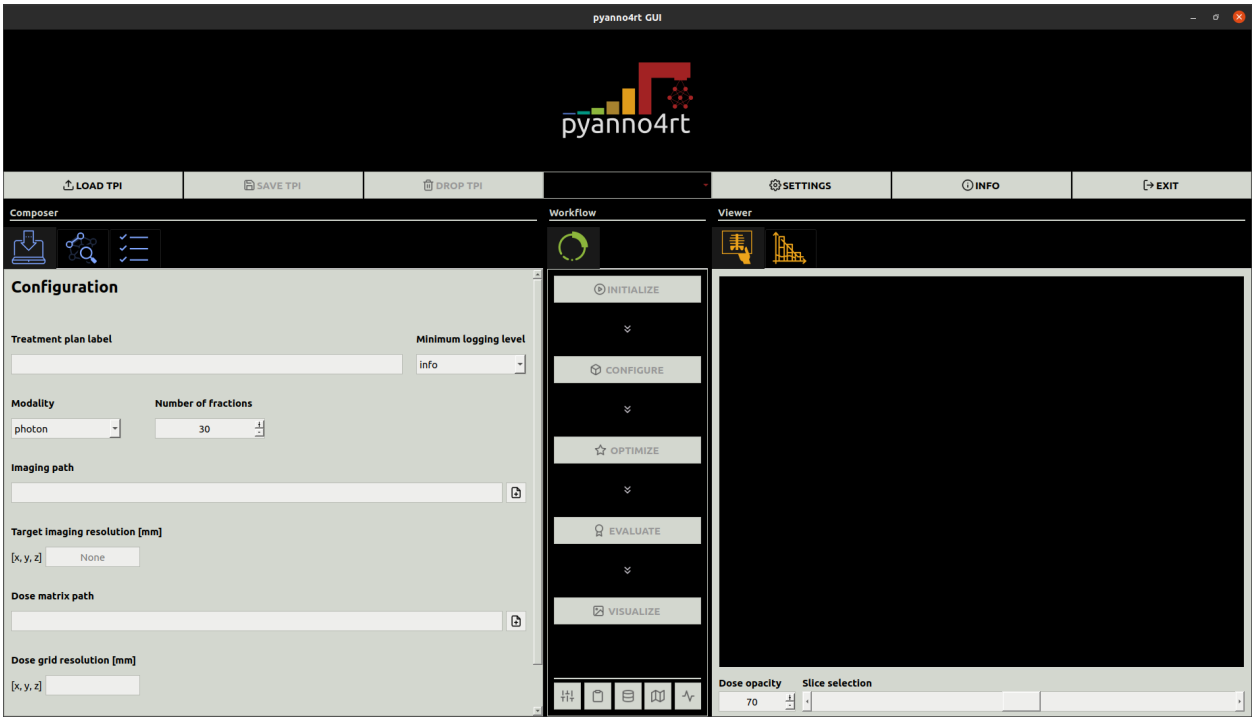
```
gui = GraphicalUserInterface()
```

6.4.2 GUI opening

Then, you can open the GUI window directly using the launch method.

```
gui.launch()
```

Below you can see the main window of the GUI that should now appear.



Without going into detail, the most important widgets shall be described here:

Alternatively, you can launch the GUI directly with an instance.

```
gui.launch(tp)
```

6.4.3 Fetching treatment plans from the GUI

The GUI has an internal dictionary in which objects of the TreatmentPlan class generated from the interface are stored. These can also be retrieved after closing the GUI using the fetch method.

```
tps_gui = gui.fetch()
```

6.5 Outro

We very much hope that this little example illustrates the basic usage of our package for treatment plan optimization. If you have any questions or suggestions, or in the (hopefully unlikely) event that something does not work, please take a look at the "Help and support" section and drop us a line. We would also be happy if you leave a positive comment or recommend our work to others. Thank you for using pyanno4rt

CHAPTER

SEVEN

TEMPLATES

7.1 Optimization components

This section covers the available optimization components (objectives and constraints). If specific user inputs are required, we set a placeholder tagged with "<>". Feel free to copy the contents of the cells for your purposes!

7.1.1 Decision Tree NTCP

```
{
    'class': 'Decision Tree NTCP',
    'parameters': {
        'model_parameters': {
            'model_label': 'decisionTreeNTCP',
            'model_folder_path': None,
            'data_path': '<your_data_path>', # Insert the path to your data file here
            'feature_filter': {
                'features': [],
                'filter_mode': 'remove'
                },
            'label_name': '<your_label>',
            'label_bounds': [1, 1],
            'time_variable_name': None,
            'label_viewpoint': 'long-term',
            'fuzzy_matching': True,
            'preprocessing_steps': ['Equalizer'],
            'tune_space': {
                'criterion': ['gini', 'entropy'],
                'splitter': ['best', 'random'],
                'max_depth': list(range(1, 21)),
                'min_samples_split': [0.0, 1.0],
                'min_samples_leaf': [0.0, 0.5],
                'min_weight_fraction_leaf': [0.0, 0.5],
                'max_features': list(range(1, 21)),
                'class_weight': [None, 'balanced'],
                'ccp_alpha': [0.0, 1.0]
                },
            'tune_evaluations': 50,
            'tune_score': 'Logloss',
            'tune_splits': 5,
```

(continues on next page)

(continued from previous page)

```
            'inspect_model': False,
            'evaluate_model': False,
            'oof_splits': 5,
            'write_features': False,
            'display_options': {
                'graphs': ['AUC-ROC', 'AUC-PR', 'F1'],
                'kpis': ['Logloss', 'Brier score',
                         'Subset accuracy', 'Cohen Kappa',
                         'Hamming loss', 'Jaccard score',
                         'Precision', 'Recall', 'F1 score',
                         'MCC', 'AUC']
                },
            },
        'embedding': 'active',
        'weight': 1,
        'link': None,
        'identifier': None
        }
    }
```

CHAPTER

EIGHT

API REFERENCE

This page contains auto-generated API reference documentation.

8.1 pyanno4rt

Python-based advanced numerical nonlinear optimization for radiotherapy (pyanno4rt) module.

---

pyanno4rt is a Python package for conventional and outcome prediction model-based inverse photon and proton treatment plan optimization, including radiobiological and machine learning (ML) models for tumor control probability (TCP) and normal tissue complication probability (NTCP).

This module aims to provide methods and classes for the import of patient data from different sources, the individual configuration and management of treatment plan instances, multi-objective treatment plan optimization, data-driven outcome prediction model handling, evaluation, and visualization.

It also features an easy-to-use and clear graphical user interface.

8.1.1 Subpackages

pyanno4rt.base

Base module.

This module aims to provide base classes to generate treatment plans.

Overview

Table 1: Classes

TreatmentPlan	Base treatment plan class.

Classes

class pyanno4rt.base.**TreatmentPlan**(*configuration*, *optimization*, *evaluation=None*)

Base treatment plan class.

This class enables configuration, optimization, evaluation, and visualization of individual IMRT treatment plans. It therefore provides a simple, but extensive interface using input dictionaries for the different parameter groups.

Parameters

- **configuration** (*dict*) – Dictionary with the treatment plan configuration parameters.
  - **label**
    [str] Unique identifier for the treatment plan.

    Note: Uniqueness of the label is important because it prevents overwriting processes between different treatment plan instances by isolating their datahubs, logging channels and general storage paths.

  - **min_log_level**
    [{'debug', 'info', 'warning', 'error, 'critical'}, default='info'] Minimum logging level.
  - **modality**
    [{'photon', 'proton'}] Treatment modality, needs to be consistent with the dose calculation inputs.

    Note: If the modality is 'photon', `DoseProjection` with neutral RBE of 1.0 is automatically applied, whereas for the modality 'proton', `ConstantRBEProjection` with constant RBE of 1.1 is used.

  - **number_of_fractions**
    [int] Number of fractions according to the treatment scheme.
  - **imaging_path**
    [str] Path to the CT and segmentation data.

    Note: It is assumed that CT and segmentation data are included in a single file (.mat or .p) or a series of files (.dcm), whose content follows the pyanno4rt data structure.

  - **target_imaging_resolution**
    [list or None, default=None] Imaging resolution for post-processing interpolation of the CT and segmentation data, only used if a list is passed.
  - **dose_matrix_path**
    [str] Path to the dose-influence matrix file (.mat or .npy).
  - **dose_resolution**
    [list] Size of the dose grid in [*mm*] per dimension, needs to be consistent with the dose calculation inputs.
- **optimization** (*dict*) – Dictionary with the treatment plan optimization parameters.
  - **components**
    [dict] Optimization components for each segment of interest, i.e., objective functions and constraints.

Note: The declaration scheme for a single component is

{: {'type': <1>, 'instance': {'class': <2>, 'parameters': <3>}

* <1>: 'objective' or 'constraint'
* <2>: component label (see note below)
* <3> parameter dictionary for the component (see the component classes for details)

Multiple objective functions or constraints can be assigned simultaneously by passing a list of class/parameter dictionaries for the 'instance' key.

The following components are currently available:

* 'Decision Tree NTCP' `DecisionTreeNTCP`
* 'Decision Tree TCP' `DecisionTreeTCP`
* 'Dose Uniformity' `DoseUniformity`
* 'Equivalent Uniform Dose' `EquivalentUniformDose`
* 'K-Nearest Neighbors NTCP' `KNeighborsNTCP`
* 'K-Nearest Neighbors TCP' `KNeighborsTCP`
* 'Logistic Regression NTCP' `LogisticRegressionNTCP`
* 'Logistic Regression TCP' `LogisticRegressionTCP`
* 'LQ Poisson TCP' `LQPoissonTCP`
* 'Lyman-Kutcher-Burman NTCP' `LymanKutcherBurmanNTCP`
* 'Maximum DVH' `MaximumDVH`
* 'Mean Dose' `MeanDose`
* 'Minimum DVH' `MinimumDVH`
* 'Naive Bayes NTCP' `NaiveBayesNTCP`
* 'Naive Bayes TCP' `NaiveBayesTCP`
* 'Neural Network NTCP' `NeuralNetworkNTCP`
* 'Neural Network TCP' `NeuralNetworkTCP`
* 'Random Forest NTCP' `RandomForestNTCP`
* 'Random Forest TCP' `RandomForestTCP`
* 'Squared Deviation' `SquaredDeviation`
* 'Squared Overdosing' `SquaredOverdosing`
* 'Squared Underdosing' `SquaredUnderdosing`
* 'Support Vector Machine NTCP' `SupportVectorMachineNTCP`
* 'Support Vector Machine TCP' `SupportVectorMachineTCP`

– **method**

[{'lexicographic', 'pareto', 'weighted-sum'}, default='weighted-sum'] Single- or multi-criteria optimization method, see the classes `LexicographicOptimization` `ParetoOptimization` `WeightedSumOptimization`.

  * 'lexicographic' : sequential optimization based on a preference order
  * 'pareto' : parallel optimization based on the criterion of pareto optimality
  * 'weighted-sum' : parallel optimization based on a weighted-sum scalarization of the objective function

- **solver**
  [{'proxmin', 'pymoo', 'pypop7', 'scipy'}, default='scipy'] Python package to be used for solving the optimization problem, see the classes `ProxminSolver` `PymooSolver` `PyPop7Solver` `SciPySolver`.

  * 'proxmin' : proximal algorithms provided by Proxmin
  * 'pymoo' : multi-objective algorithms provided by Pymoo
  * 'pypop7': population-based algorithms provided by PyPop7
  * 'scipy' : local algorithms provided by SciPy

  Note: The 'pareto' method currently only works with the 'pymoo' solver option. Constraints are not supported by the 'pypop7' solver option.

- **algorithm**
  [str] Solution algorithm from the chosen solver:

  * solver='proxmin' : {'admm', 'pgm', 'sdmm'}, default='pgm'
    · 'admm' : alternating direction method of multipliers
    · 'pgm' : proximal gradient method
    · 'sdmm' : simultaneous direction method of multipliers
  * solver='pymoo' : {'NSGA3'}, default='NSGA3'
    · 'NSGA3' : non-dominated sorting genetic algorithm III
  * solver='pypop7' : {'LMCMA', 'LMMAES'}, default='LMCMA'
    · 'LMCMA' : limited-memory covariance matrix adaptation
    · 'LMMAES' : limited-memory matrix adaptation evolution strategy
  * solver='scipy' : {'L-BFGS-B', 'TNC', 'trust-constr'}, default='L-BFGS-B'
    · 'L-BFGS-B' : bounded limited memory Broyden-Fletcher-Goldfarb-Shanno method
    · 'TNC' : truncated Newton method
    · 'trust-constr' : trust-region constrained method

  Note: Constraints are supported by all algorithms except the 'L-BFGS-B' algorithm.

- **initial_strategy**
  [{'data-medoid', 'target-coverage', 'warm-start'}, default='target-coverage'] Initialization strategy for the fluence vector (see the class `FluenceInitializer`).

  * 'data-medoid' : fluence vector initialization with respect to data medoid points
  * 'target-coverage' : fluence vector initialization with respect to tumor coverage

* 'warm-start' : fluence vector initialization with respect to a reference optimal point

Note: Data-medoid initialization works best for a single dataset or multiple datasets with a high degree of similarity. Otherwise, the initial fluence vector may lose its individual representativeness.

– **initial_fluence_vector**
[list or None, default=None] User-defined initial fluence vector for the optimization problem, only used if initial_strategy='warm-start' (see the class `FluenceInitializer`).

– **lower_variable_bounds**
[int, float, list or None, default=0] Lower bound(s) on the decision variables.

– **upper_variable_bounds**
[int, float, list or None, default=None] Upper bound(s) on the decision variables.

Note: There are two options to set lower and upper bounds for the variables:

1) Passing a single numeric value translates into uniform bounds across all variables (where None for the lower and/or upper bound indicates infinity bounds)

2) Passing a list translates into non-uniform bounds (here, the length of the list needs to be equal to the number of decision variables)

– **max_iter**
[int, default=500] Maximum number of iterations taken for the solver to converge.

– **tolerance**
[float, default=1e-3] Precision goal for the objective function value.

- `evaluation` (*dict, default={}*) – Dictionary with the treatment plan evaluation parameters.

  – **dvh_type**
  [{'cumulative', 'differential'}, default=cumulative'] Type of DVH to be evaluated.

  – **number_of_points**
  [int, default=1000] Number of (evenly-spaced) points for which to evaluate the DVH.

  – **reference_volume**
  [list, default=[2, 5, 50, 95, 98]] Reference volumes for which to evaluate the inverse DVH values.

  – **reference_dose**
  [list, default=[]] Reference dose values for which to evaluate the DVH values.

  Note: If the default value [] is used, reference dose levels will be determined automatically.

  – **display_segments**
  [list, default=[]] Names of the segmented structures to be displayed.

Note: If the default value [] is used, all segments will be displayed.

– **display_metrics**
[list, default=[]] Names of the plan evaluation metrics to be displayed.

Note: If the default value [] is used, all metrics will be displayed.

The following metrics are currently available:

* 'mean': mean dose
* 'std': standard deviation of the dose
* 'max': maximum dose
* 'min': minimum dose
* 'Dx': dose quantile(s) for level x (reference_volume)
* 'Vx': volume quantile(s) for level x (reference_dose)
* 'CI': conformity index
* 'HI': homogeneity index

`configuration`
See 'Parameters'.

Type
dict

`optimization`
See 'Parameters'.

Type
dict

`evaluation`
See 'Parameters'.

Type
dict

`input_checker`
The object used to approve the input dictionaries.

Type
object of class `InputChecker`

`logger`
The internal object used to print and store logging messages.

Type
object of class `Logger`

`datahub`
The object used to manage and distribute information units.

Type
object of class `Datahub`

patient_loader

The object used to import and type-convert CT and segmentation data.

Type
object of class `PatientLoader`

plan_generator

The object used to set and type-convert plan properties.

Type
object of class `PlanGenerator`

dose_info_generator

The object used to specify and type-convert dose (grid) properties.

Type
object of class `DoseInfoGenerator`

fluence_optimizer

The object used to solve the fluence optimization problem.

Type
object of class `FluenceOptimizer`

dose_histogram

The object used to evaluate the dose-volume histogram (DVH).

Type
object of class `DVHEvaluator`

dosimetrics

The object used to evaluate the dosimetrics.

Type
object of class `DosimetricsEvaluator`

visualizer

The object used to visualize the treatment plan.

Type
object of class `Visualizer`

Example

Our Read the Docs page (https://pyanno4rt.readthedocs.io/en/latest/) features a step-by-step example for the application of this class. You will also find code templates there, e.g. for the components.

Overview

Table 2: Methods

configure()	Initialize the configuration classes and process the input data.
optimize()	Initialize the fluence optimizer and solve the problem.
evaluate()	Initialize the evaluation classes and compute the plan metrics.
visualize(parent)	Initialize the visualization interface and launch it.
compose()	Compose the treatment plan by cycling the entire workflow.
update(key_value_pairs)	Update the input dictionaries by specific key-value pairs.

Members

configure()

Initialize the configuration classes and process the input data.

optimize()

Initialize the fluence optimizer and solve the problem.

Raises

AttributeError – If the treatment plan has not been configured yet.

evaluate()

Initialize the evaluation classes and compute the plan metrics.

Raises

AttributeError – If the treatment plan has not been optimized yet.

visualize(*parent=None*)

Initialize the visualization interface and launch it.

Parameters

parent (object of class `MainWindow`, default=None) – The (optional) object used as a parent window for the visualization interface.

Raises

AttributeError – If the treatment plan has not been optimized (and evaluated) yet.

compose()

Compose the treatment plan by cycling the entire workflow.

update(*key_value_pairs*)

Update the input dictionaries by specific key-value pairs.

Parameters

key_value_pairs (*dict*) – Dictionary with the keys and values to update.

Raises

KeyError – If any update key is not included in the parameter dictionaries.

pyanno4rt.datahub

Datahub module.

This module aims to provide methods and classes to centralize and distribute information units within each treatment plan.

Overview

Table 3: Classes

Datahub	Central data storage and management hub class.

Classes

class pyanno4rt.datahub.**Datahub**(**args*)

Central data storage and management hub class.

This class provides a singleton datahub for centralizing the information units generated across one or multiple treatment plans, e.g. dictionaries with CT and segmentation data, to efficiently manage and distribute them.

Parameters

***args** (*tuple*) – Tuple with optional (non-keyworded) parameters. The element args[0] refers to the treatment plan label, while args[1] is a `Logger` object and args[2] is an `InputChecker` object. Only required for (re-)instantiating a datahub.

instances

Dictionary with pairs of treatment plan labels and associated `Datahub` objects.

Type
dict

label

Label of the current active treatment plan instance.

Type
str

input_checker

The object used to approve the input dictionaries.

Type
object of class `InputChecker`

logger

The object used to print and store logging messages.

Type
object of class `Logger`

computed_tomography

Dictionary with information on the CT images.

Type
dict

segmentation

Dictionary with information on the segmented structures.

Type
dict

plan_configuration

Dictionary with information on the plan configuration.

Type
dict

dose_information

Dictionary with information on the dose grid.

Type
dict

`optimization`

Dictionary with information on the fluence optimization.

Type
dict

`datasets`

Dictionary with pairs of model labels and associated external datasets used for model fitting. Each dataset is a dictionary itself, holding information on the raw data and the features/labels.

Type
dict

`feature_maps`

Dictionary with pairs of model labels and associated feature maps. Each feature map holds links between the features from the respective dataset, the segments, and the definitions from the feature catalogue.

Type
dict

`model_instances`

Dictionary with pairs of model labels and associated model instances, i.e., the prediction model, the model configuration dictionary, and the model hyperparameters obtained from hyperparameter tuning.

Type
dict

`model_inspections`

Dictionary with pairs of model labels and associated model inspectors. Each inspector holds information on the inspection measures calculated.

Type
dict

`model_evaluations`

Dictionary with pairs of model labels and associated model evaluators. Each evaluator holds information on the evaluation measures calculated.

Type
dict

`dose_histogram`

Dictionary with information on the cumulative or differential dose-volume histogram for each segmented structure.

Type
dict

`dosimetrics`

Dictionary with information on the dosimetrics for each segmented structure.

Type
dict

Overview

Table 4: Attributes

instances	-
label	-
input_checker	-
logger	-
computed_tomography	-
segmentation	-
plan_configuration	-
dose_information	-
optimization	-
datasets	-
feature_maps	-
model_instances	-
model_inspections	-
model_evaluations	-
dose_histogram	-
dosimetrics	-

Members

instances

label

input_checker

logger

computed_tomography

segmentation

plan_configuration

dose_information

optimization

datasets

feature_maps

model_instances

model_inspections

model_evaluations

dose_histogram

dosimetrics

pyanno4rt.dose_info

Dose information module.

This module aims to provide methods and classes to generate the dose information dictionary.

Overview

Table 5: Classes

DoseInfoGenerator	Dose information generation class.

Classes

class pyanno4rt.dose_info.**DoseInfoGenerator**(*number_of_fractions*, *dose_matrix_path*, *dose_resolution*)

Dose information generation class.

This class provides methods to generate the dose information dictionary for the management and retrieval of dose grid properties and dose-related parameters.

Parameters

- **dose_resolution** (*list*) – Size of the dose grid in [*mm*] per dimension.
- **number_of_fractions** (*int*) – Number of fractions according to the treatment scheme.
- **dose_matrix_path** (*str*) – Path to the dose-influence matrix file (.mat or .npy).

number_of_fractions

See 'Parameters'.

Type
int

dose_matrix_path

See 'Parameters'.

Type
str

dose_resolution

See 'Parameters'.

Type
tuple

Overview

Table 6: Methods

generate()	Generate the dose information dictionary.

Members

generate()

Generate the dose information dictionary.

pyanno4rt.evaluation

Treatment plan evaluation module.

This module aims to provide methods and classes to evaluate the generated treatment plans.

Overview

Table 7: Classes

DosimetricsEvaluator	Dosimetrics evaluation class.
DVHEvaluator	DVH evaluation class.

Classes

class pyanno4rt.evaluation.**DosimetricsEvaluator**(*reference_volume*, *reference_dose*, *display_segments*, *display_metrics*)

Dosimetrics evaluation class.

This class provides methods to evaluate dosimetrics as a means to quantify dose distributions from a treatment plan across the segments. Dosimetrics include statistical location and dispersion measures, DVH indicators as well as conformity (CI) and homogeneity index (HI).

Parameters

- **reference_volume** (*list*) – Reference volumes for which to evaluate the inverse DVH indicators.
- **reference_dose** (*list*) – Reference dose values for which to evaluate the DVH indicators.
- **display_segments** (*list*) – Names of the segmented structures to be displayed.
- **display_metrics** (*list*) – Names of the metrics to be displayed.

reference_volume

See 'Parameters'.

Type

tuple

reference_dose

See 'Parameters'.

Type
tuple

display_segments

See 'Parameters'.

Type
tuple

display_metrics

See 'Parameters'.

Type
tuple

Overview

Table 8: Methods

evaluate(dose_cube)	Evaluate the dosimetrics for all segments.

Members

evaluate(*dose_cube*)

Evaluate the dosimetrics for all segments.

Parameters
dose_cube (*ndarray*) – Three-dimensional array with the dose values (CT resolution).

class pyanno4rt.evaluation.**DVHEvaluator**(*dvh_type*, *number_of_points*, *display_segments*)

DVH evaluation class.

This class provides methods to evaluate dose-volume histograms (DVH) as a means to quantify dose distributions from a treatment plan across the segments. Both cumulative and differential DVH can be evaluated.

Parameters

- **dvh_type** (*{'cumulative', 'differential'}*) – Type of DVH to be evaluated.
- **number_of_points** (*int*) – Number of (evenly-spaced) points for which to evaluate the DVH.
- **display_segments** (*list*) – Names of the segmented structures to be displayed.

dvh_type

See 'Parameters'.

Type
{'cumulative', 'differential'}

number_of_points

See 'Parameters'.

Type
int

display_segments

See 'Parameters'.

Type

tuple

Overview

Table 9: Methods

evaluate(dose_cube)	Evaluate the DVH for all segments.

Members

evaluate(*dose_cube*)

Evaluate the DVH for all segments.

Parameters

dose_cube (*ndarray*) – Three-dimensional array with the dose values (CT resolution).

pyanno4rt.gui

Graphical user interface module.

The module aims to provide methods and classes to …

Subpackages

pyanno4rt.gui.custom_widgets

Custom widgets module.

The module aims to provide methods and classes to …

Overview

Table 10: Classes

DVHWidget	.
SliceWidget	.

Classes

class pyanno4rt.gui.custom_widgets.**DVHWidget**(*parent=None*)

Bases: PyQt5.QtWidgets.QWidget

.

Overview

Table 11: Methods

add_style_and_data(dose_histogram)	.
get_segment_statistics(event)	.
reset_dvh()	.
select_dvh_curve(event)	.
update_crosshair(event)	Update the crosshair at mouse moves.
update_dvh()	.

Members

add_style_and_data(*dose_histogram*)

.

get_segment_statistics(*event*)

.

reset_dvh()

.

select_dvh_curve(*event*)

.

update_crosshair(*event*)

Update the crosshair at mouse moves.

update_dvh()

.

class pyanno4rt.gui.custom_widgets.**SliceWidget**(*parent=None*)

Bases: PyQt5.QtWidgets.QWidget

.

Overview

Table 12: Methods

add_ct(ct_cube)	.
add_dose(dose_cube)	-
add_segments(computed_tomography, segmentation)	-
change_dose_opacity()	.
change_image_slice()	.
reset_images()	.
update_images()	Update the images when scrolling.

Members

add_ct(*ct_cube*)

.

add_dose(*dose_cube*)

add_segments(*computed_tomography*, *segmentation*)

change_dose_opacity()

.

change_image_slice()

.

reset_images()

.

update_images()

Update the images when scrolling.

pyanno4rt.gui.windows

GUI windows module.

The module aims to provide methods and classes to …

Overview

Table 13: Classes

InfoWindow	Information window for the GUI.
LogWindow	Logging window for the application.
PlanCreationWindow	Plan creation window for the application.
SettingsWindow	Settings window for the GUI.
TextWindow	Text window for the application.
TreeWindow	Tree window for the application.
MainWindow	Main window for the GUI.

Classes

class pyanno4rt.gui.windows.**InfoWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, pyanno4rt.gui.compilations.info_window.Ui_info_window

Information window for the GUI.

This class creates the information window for the graphical user interface, including some general information on the package.

Overview

Table 14: Methods

position()	.
close()	.

Members

position()

.

close()

.

class pyanno4rt.gui.windows.**LogWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, pyanno4rt.gui.compilations.log_window.Ui_log_window

Logging window for the application.

This class creates the log window for the graphical user interface, including the output of the logger.

Overview

Table 15: Methods

position()	.
update_log_output()	.
close()	.

Members

position()

.

update_log_output()

.

close()

.

class pyanno4rt.gui.windows.**PlanCreationWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, pyanno4rt.gui.compilations.plan_creation_window.Ui_plan_create_window

Plan creation window for the application.

This class creates a plan creation window for the graphical user interface, including input fields to declare a plan.

Overview

Table 16: Methods

position()	.
update_by_new_plan_label()	.
update_by_new_plan_reference()	.
add_imaging_path()	Add the CT and segmentation data from a folder.
add_dose_matrix_path()	Add the dose-influence matrix from a folder.
create()	.
close()	.

Members

position()

.

update_by_new_plan_label()

.

update_by_new_plan_reference()

.

add_imaging_path()

Add the CT and segmentation data from a folder.

add_dose_matrix_path()

Add the dose-influence matrix from a folder.

create()

.

close()

.

class pyanno4rt.gui.windows.**SettingsWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, pyanno4rt.gui.compilations.settings_window.Ui_settings_window

Settings window for the GUI.

This class creates the settings window for the graphical user interface, including some user-definable parameters.

Overview

Table 17: Methods

position()	.
get_fields()	.
set_fields(settings)	.
reset()	.
save_apply_close()	.

Members

position()

.

get_fields()

.

set_fields(*settings*)

.

reset()

.

save_apply_close()

.

class pyanno4rt.gui.windows.**TextWindow**(*parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, pyanno4rt.gui.compilations.text_window.Ui_text_window

Text window for the application.

This class creates a text window for the graphical user interface, including a scrollable text box for display.

Overview

Table 18: Methods

position()	.
close()	.

Members

position()

.

close()

.

class pyanno4rt.gui.windows.**TreeWindow**(*title*, *parent=None*)

Bases: PyQt5.QtWidgets.QMainWindow, pyanno4rt.gui.compilations.tree_window.Ui_tree_window

Tree window for the application.

This class creates a tree window for the graphical user interface, including a tree-based table view for dictionaries.

Overview

Table 19: Methods

position()	.
create_tree_from_dict(data, parent)	.
show_item_text(tree, item)	.
expand_all()	.
collapse_all()	.
close()	.

Members

position()

.

create_tree_from_dict(*data=None*, *parent=None*)

.

show_item_text(*tree*, *item*)

.

expand_all()

.

collapse_all()

.

close()

.

class pyanno4rt.gui.windows.**MainWindow**(*treatment_plan*, *application=None*)

Bases: PyQt5.QtWidgets.QMainWindow, pyanno4rt.gui.compilations.main_window.Ui_main_window

Main window for the GUI.

This class creates the main window for the graphical user interface, including logo, labels, and input/control elements.

Parameters

- **treatment_plan** (object of class *TreatmentPlan*) – Instance of the class *TreatmentPlan*, which provides methods and classes to generate treatment plans.
- **application** (object of class *SpyderQApplication*) – Instance of the class *SpyderQApplication* for managing control flow and main settings of the graphical user interface.

Overview

Table 20: Methods

connect_signals()	Connect the event signals to the GUI elements.
eventFilter(source, event)	Customize the event filters.
set_initial_plan(treatment_plan)	Set the initial treatment plan in the GUI.
set_enabled(fieldnames)	Enable multiple fields by their names.
set_disabled(fieldnames)	Disable multiple fields by their names.
set_zero_line_cursor(fieldnames)	Set the line edit cursor positions to zero.
set_styles(key_value_pairs)	Set the element stylesheets from key-value pairs.
activate(treatment_plan)	Activate a treatment plan instance in the GUI.
load_tpi()	Load the treatment plan from a snapshot folder.
save_tpi()	Save the treatment plan to a snapshot folder.
drop_tpi()	Remove the current treatment plan.
select_plan()	Select a treatment plan.
initialize()	Initialize the treatment plan.
configure()	Configure the treatment plan.
optimize()	Optimize the treatment plan.
evaluate()	Evaluate the treatment plan.
visualize()	Visualize the treatment plan.
update_configuration()	Update the configuration parameters.
update_optimization()	Update the optimization parameters.
update_evaluation()	Update the evaluation parameters.
set_configuration()	Set the configuration parameters.
set_optimization()	Set the optimization parameters.
set_evaluation()	Set the evaluation parameters.
clear_configuration()	Clear the configuration parameters.
clear_optimization()	Clear the optimization parameters.
clear_evaluation()	Clear the evaluation parameters.
transform_configuration_to_dict()	Transform the configuration fields into a dictionary.
transform_optimization_to_dict()	Transform the optimization fields into a dictionary.
transform_evaluation_to_dict()	Transform the evaluation fields into a dictionary.
open_plan_creation_window()	Open the plan creation window.
open_settings_window()	Open the settings window.
open_info_window()	Open the information window.
exit_window()	Exit the session and close the window.
open_parameter_window()	Open the plan parameter window.
open_plan_window()	Open the plan data window.
open_model_data_window()	Open the model data window.
open_feature_map_window()	Open the feature map window.
open_log_window()	Open the log window.
open_question_dialog()	Open a question dialog.
add_imaging_path()	Add the CT and segmentation data from a folder.
add_dose_matrix_path()	Add the dose-influence matrix from a folder.
remove_component()	Remove the selected component from the instance.
add_initial_fluence_vector()	Add the initial fluence vector from a file.
add_lower_var_bounds()	Add the lower variable bounds from a file.
add_upper_var_bounds()	Add the upper variable bounds from a file.
update_by_plan_label()	.
update_by_initial_strategy()	Update the GUI by the initial strategy.
update_by_initial_fluence()	Update the GUI by the initial fluence vector.
update_by_reference()	Update the GUI by the reference plan.

continues on next page

Table 20 – continued from previous page

update_by_method()	Update the GUI by the optimization method.
update_by_solver()	Update the GUI by the solver.
update_reference_plans()	Update the available reference plans.

Members

connect_signals()

Connect the event signals to the GUI elements.

eventFilter(*source*, *event*)

Customize the event filters.

Parameters

- **source** – ...
- **event** – ...

Return type

...

set_initial_plan(*treatment_plan*)

Set the initial treatment plan in the GUI.

Parameters
treatment_plan – ...

set_enabled(*fieldnames*)

Enable multiple fields by their names.

Parameters
fieldnames – ...

set_disabled(*fieldnames*)

Disable multiple fields by their names.

Parameters
fieldnames – ...

set_zero_line_cursor(*fieldnames*)

Set the line edit cursor positions to zero.

Parameters
fieldnames – ...

set_styles(*key_value_pairs*)

Set the element stylesheets from key-value pairs.

Parameters
key_value_pairs – ...

activate(*treatment_plan*)

Activate a treatment plan instance in the GUI.

Parameters
treatment_plan – ...

`load_tpi()`

Load the treatment plan from a snapshot folder.

`save_tpi()`

Save the treatment plan to a snapshot folder.

`drop_tpi()`

Remove the current treatment plan.

`select_plan()`

Select a treatment plan.

`initialize()`

Initialize the treatment plan.

Returns
Indicator for the success of the initialization.

Return type
bool

`configure()`

Configure the treatment plan.

Returns
Indicator for the success of the configuration.

Return type
bool

`optimize()`

Optimize the treatment plan.

Returns
Indicator for the success of the optimization.

Return type
bool

`evaluate()`

Evaluate the treatment plan.

Returns
Indicator for the success of the evaluation.

Return type
bool

`visualize()`

Visualize the treatment plan.

Returns
Indicator for the success of the visualization.

Return type
bool

`update_configuration()`

Update the configuration parameters.

update_optimization()

Update the optimization parameters.

update_evaluation()

Update the evaluation parameters.

set_configuration()

Set the configuration parameters.

set_optimization()

Set the optimization parameters.

set_evaluation()

Set the evaluation parameters.

clear_configuration()

Clear the configuration parameters.

clear_optimization()

Clear the optimization parameters.

clear_evaluation()

Clear the evaluation parameters.

transform_configuration_to_dict()

Transform the configuration fields into a dictionary.

Returns
Dictionary with the configuration parameters.

Return type
dict

transform_optimization_to_dict()

Transform the optimization fields into a dictionary.

Returns
Dictionary with the optimization parameters.

Return type
dict

transform_evaluation_to_dict()

Transform the evaluation fields into a dictionary.

Returns
Dictionary with the evaluation parameters.

Return type
dict

open_plan_creation_window()

Open the plan creation window.

open_settings_window()

Open the settings window.

open_info_window()

Open the information window.

exit_window()

Exit the session and close the window.

open_parameter_window()

Open the plan parameter window.

open_plan_window()

Open the plan data window.

open_model_data_window()

Open the model data window.

open_feature_map_window()

Open the feature map window.

open_log_window()

Open the log window.

open_question_dialog()

Open a question dialog.

add_imaging_path()

Add the CT and segmentation data from a folder.

add_dose_matrix_path()

Add the dose-influence matrix from a folder.

remove_component()

Remove the selected component from the instance.

add_initial_fluence_vector()

Add the initial fluence vector from a file.

add_lower_var_bounds()

Add the lower variable bounds from a file.

add_upper_var_bounds()

Add the upper variable bounds from a file.

update_by_plan_label()

.

update_by_initial_strategy()

Update the GUI by the initial strategy.

update_by_initial_fluence()

Update the GUI by the initial fluence vector.

update_by_reference()

Update the GUI by the reference plan.

update_by_method()

Update the GUI by the optimization method.

update_by_solver()

Update the GUI by the solver.

update_reference_plans()

Update the available reference plans.

Overview

Table 21: Classes

GraphicalUserInterface	Graphical user interface class.

Classes

class pyanno4rt.gui.**GraphicalUserInterface**

Graphical user interface class.

This class provides …

Parameters

...

...

Overview

Table 22: Methods

launch(plan)	Launch the graphical user interface.
fetch()	Get the treatment plan dictionary of the GUI.
closeEvent(event)	Close the application.

Members

launch(*plan=None*)

Launch the graphical user interface.

fetch()

Get the treatment plan dictionary of the GUI.

closeEvent(*event*)

Close the application.

Parameters

event (object of class *QCloseEvent*) – Instance of the class *QCloseEvent* to be triggered at window closing.

pyanno4rt.input_check

Input checking module.

This module aims to provide classes and functions to perform input parameter checks.

Subpackages

pyanno4rt.input_check.check_functions

Check functions module.

This module aims to provide a collection of basic validity check functions.

Overview

Table 23: Function

check_components(label, data, check_functions)	Check the optimization components.
check_dose_matrix(dose_shape, dose_matrix_rows)	Check the equality between the number of dose voxels calculated from the dose resolution inputs and implied by the dose-influence matrix.
check_feature_filter(label, data, check_functions)	Check the feature filter.
check_key_in_dict(label, data, keys)	Check if a key is not featured in a dictionary.
check_length(label, data, reference, sign)	Check if the length of a vector-type object is invalid.
check_path(label, data)	Check if a file or directory path is invalid.
check_regular_extension(label, data, extensions)	Check if a file path is irregular or has an invalid extension.
check_regular_extension_direc data, extensions)	Check if a directory path is irregular or has invalid file extensions.
check_subtype(label, data, types)	Check if any element type in a list or tuple is invalid.
check_type(label, data, types, type_condition)	Check if the input data type is invalid.
check_value(label, data, reference, sign, is_vector)	Check if the data has an invalid value range.
check_value_in_set(label, data, options, value_condition)	Check if a value is not included in a set of options.

Functions

pyanno4rt.input_check.check_functions.**check_components**(*label*, *data*, *check_functions*)

Check the optimization components.

Parameters

- **label** (*str*) – Label for the item to be checked ('components').
- **data** (*dict*) – Dictionary with the optimization components.
- **check_functions** (*tuple*) – Tuple with the individual check functions for the dictionary items.

pyanno4rt.input_check.check_functions.**check_dose_matrix**(*dose_shape*, *dose_matrix_rows*)

Check the equality between the number of dose voxels calculated from the dose resolution inputs and implied by the dose-influence matrix.

Parameters

- **dose_shape** (*tuple*) – Tuple with the number of dose grid points per axis, calculated from the dose resolution inputs.
- **dose_matrix_rows** (*int*) – Number of rows in the dose-influence matrix (the number of voxels in the dose grid).

Raises

ValueError – If the product of the elements in dose_shape is not equal to the value of dose_matrix_rows.

pyanno4rt.input_check.check_functions.**check_feature_filter**(*label*, *data*, *check_functions*)

Check the feature filter.

Parameters

- **label** (*str*) – Label for the item to be checked ('feature_filter').
- **data** (*dict*) – Dictionary with the parameters of the feature filter.
- **check_functions** (*tuple*) – Tuple with the individual check functions for the dictionary items.

pyanno4rt.input_check.check_functions.**check_key_in_dict**(*label*, *data*, *keys*)

Check if a key is not featured in a dictionary.

Parameters

- **key** (*str*) – Label for the item to be checked.
- **data** (*dict*) – Dictionary with the reference keys.
- **keys** (*tuple*) – Tuple with the keys to search for in the dictionary.

Raises

KeyError – If a key is not featured in the dictionary.

pyanno4rt.input_check.check_functions.**check_length**(*label*, *data*, *reference*, *sign*)

Check if the length of a vector-type object is invalid.

Parameters

- **label** (*str*) – Label for the item to be checked.
- **data** (*list, tuple or ndarray*) – Vector-type object with length property.
- **reference** (*int*) – Reference value for the length comparison.
- **sign** (*{'==', '>', '>=', '<', '<='}*) – Sign for the length comparison.

Raises

ValueError – If the vector-type object has an invalid length.

pyanno4rt.input_check.check_functions.**check_path**(*label*, *data*)

Check if a file or directory path is invalid.

Parameters

- **label** (*str*) – Label for the item to be checked.
- **data** (*str*) – Path to the file or directory.

Raises

IOError – If the path references an invalid file or directory.

`pyanno4rt.input_check.check_functions.`**`check_regular_extension`**(*label*, *data*, *extensions*)

Check if a file path is irregular or has an invalid extension.

Parameters

- **label** (*str*) – Label for the item to be checked.
- **data** (*str*) – Path to the file.
- **extensions** (*tuple*) – Tuple with the allowed extensions for the file path.

Raises

- **FileNotFoundError** – If the path references an irregular file.
- **TypeError** – If the path has an invalid extension.

`pyanno4rt.input_check.check_functions.`**`check_regular_extension_directory`**(*label*, *data*, *extensions*)

Check if a directory path is irregular or has invalid file extensions.

Parameters

- **label** (*str*) – Label for the item to be checked.
- **data** (*str*) – Path to the file directory.
- **extensions** (*tuple*) – Tuple with the allowed extensions for the directory files.

Raises

- **NotADirectoryError** – If the path references an irregular directory.
- **TypeError** – If a file in the directory has an invalid extension.

`pyanno4rt.input_check.check_functions.`**`check_subtype`**(*label*, *data*, *types*)

Check if any element type in a list or tuple is invalid.

Parameters

- **label** (*str*) – Label for the item to be checked.
- **data** (*list or tuple*) – List or tuple with the element types to be checked.
- **types** (*type or tuple*) – Single type or tuple with the allowed element types.

Raises

TypeError – If one or more elements of the data have an invalid type.

`pyanno4rt.input_check.check_functions.`**`check_type`**(*label*, *data*, *types*, *type_condition=None*)

Check if the input data type is invalid.

Parameters

- **label** (*str*) – Label for the item to be checked.
- **data** – Input data with arbitrary type to be checked.
- **types** (*tuple or dict*) – Tuple or dictionary with the allowed data types.
- **type_condition** (*str*) – Value of the conditional variable (used as a selector if types is a dictionary).

Raises

TypeError – If the input data has an invalid type.

`pyanno4rt.input_check.check_functions.`**`check_value`**(*label*, *data*, *reference*, *sign*, *is_vector=False*)

Check if the data has an invalid value range.

Parameters

- **label** (*str*) – Label for the item to be checked.
- **data** (*int, float, None, list or tuple*) – Scalar or vector input to be checked.
- **reference** (*int or float*) – Reference for the value comparison.
- **sign** (*{'==', '>', '>=', '<', '<='}*) – Sign for the value comparison.
- **is_vector** (*bool, default=False*) – Indicator for the vector property of the data.

Raises

ValueError – If the data has an invalid value range.

pyanno4rt.input_check.check_functions.**check_value_in_set**(*label*, *data*, *options*, *value_condition=None*)

Check if a value is not included in a set of options.

Parameters

- **label** (*str*) – Label for the item to be checked.
- **data** (*str or list*) – Input value to be checked.
- **options** (*tuple or dict*) – Tuple or dictionary with the value options.
- **value_condition** (*str*) – Value of the conditional variable (used as a selector if options is a dictionary).

Raises

ValueError – If the data has a value not included in the set of options.

pyanno4rt.input_check.check_maps

Check maps module.

This module aims to provide scripts with mappings between the members of different input parameter groups and their validity check functions.

Overview

Attributes

pyanno4rt.input_check.check_maps.**component_map**

pyanno4rt.input_check.check_maps.**configuration_map**

pyanno4rt.input_check.check_maps.**evaluation_map**

pyanno4rt.input_check.check_maps.**model_display_map**

pyanno4rt.input_check.check_maps.**model_map**

pyanno4rt.input_check.check_maps.**optimization_map**

pyanno4rt.input_check.check_maps.**top_level_map**

pyanno4rt.input_check.check_maps.**tune_space_map**

Overview

Table 24: Classes

InputChecker	Input checker class.

Classes

class pyanno4rt.input_check.**InputChecker**

Input checker class.

This class provides methods to perform input checks on the user-defined parameters for objects of any class from *base*. It ensures the validity of the internal program steps with regard to the exogenous variables.

check_map

Dictionary with all mappings between parameter names and validity check functions.

Type
dict

Raises
ValueError – If non-unique parameter names are found.

Notes

The `InputChecker` class relies on the uniqueness of the parameter names to create a dictionary-based mapping. Hence, make sure to assign unique labels for all parameters to be checked!

Overview

Table 25: Methods

approve(input_dictio	Approve the input dictionary items (parameter names and values) by running the corresponding check functions.

Members

approve(*input_dictionary*)

Approve the input dictionary items (parameter names and values) by running the corresponding check functions.

Parameters
input_dictionary (*dict*) – Dictionary with the mappings between parameter names and values to be checked.

pyanno4rt.learning_model

Learning model module.

The module aims to provide methods and classes for data handling, preprocessing, learning model fitting, inspection & evaluation.

Subpackages

pyanno4rt.learning_model.dataset

Dataset module.

The module aims to provide methods and classes to import and restructure different types of learning model datasets (tabular, image-based, . . .).

Overview

Table 26: Classes

TabularDataGenerator	Tabular dataset generation class.

Classes

class pyanno4rt.learning_model.dataset.**TabularDataGenerator**(*model_label*, *feature_filter*, *label_name*, *label_bounds*, *time_variable_name*, *label_viewpoint*)

Tabular dataset generation class.

This class provides methods to load, decompose, modulate and binarize a tabular base dataset.

Parameters

- **model_label** (*str*) – Label for the machine learning model.
- **feature_filter** (*dict*) – Dictionary with a list of feature names and a value from {'retain', 'remove'} as an indicator for retaining/removing the features prior to model fitting.
- **label_name** (*str*) – Name of the label variable.
- **label_bounds** (*list*) – Bounds for the label values to binarize into positive (value lies inside the bounds) and negative class (value lies outside the bounds).
- **time_variable_name** (*str*) – Name of the time-after-radiotherapy variable (unit should be days).
- **label_viewpoint** (*{'early', 'late', 'long-term', 'longitudinal', 'profile'}*) – Time of observation for the presence of tumor control and/or normal tissue complication events.

model_label

See 'Parameters'.

Type
str

feature_filter

See 'Parameters'.

Type
dict

label_name

See 'Parameters'.

Type
str

label_bounds

See 'Parameters'.

Type
list

time_variable_name

See 'Parameters'.

Type
str

label_viewpoint

See 'Parameters'.

Type
{'early', 'late', 'long-term', 'longitudinal', 'profile'}

Overview

Table 27: Methods

generate(data_path)	Generate the data information.
decompose(dataset, feature_filter, label_name, time_variable_name)	Decompose the base tabular dataset.
modulate(data_information, label_viewpoint)	Modulate the data information.
binarize(data_information, label_bounds)	Binarize the data information.

Members

generate(*data_path*)

Generate the data information.

Parameters
data_path (*str*) – Path to the data set used for fitting the machine learning model.

Returns
Dictionary with the decomposed, modulated and binarized data information.

Return type
dict

`decompose`(*dataset*, *feature_filter*, *label_name*, *time_variable_name*)

Decompose the base tabular dataset.

Parameters

- **dataset** (`DataFrame`) – Dataframe with the feature and label names/values.
- **feature_filter** (*dict*) – Dictionary with a list of feature names and a value from {'retain', 'remove'} as an indicator for retaining/removing the features prior to model fitting.
- **label_name** (*str*) – Name of the label variable.
- **time_variable_name** (*str*) – Name of the time-after-radiotherapy variable (unit should be days).

Returns
Dictionary with the decomposed data information.

Return type
dict

`modulate`(*data_information*, *label_viewpoint*)

Modulate the data information.

Parameters

- **data_information** (*dict*) – Dictionary with the decomposed data information.
- **label_viewpoint** (*{'early', 'late', 'long-term', 'longitudinal', 'profile'}*) – Time of observation for the presence of tumor control and/or normal tissue complication events.

Returns
Dictionary with the modulated data information.

Return type
dict

`binarize`(*data_information*, *label_bounds*)

Binarize the data information.

Parameters

- **data_information** (*dict*) – Dictionary with the decomposed data information.
- **label_bounds** (*list*) – Bounds for the label values to binarize into positive (value lies inside the bounds) and negative class (value lies outside the bounds).

Returns
Dictionary with the binarized data information.

Return type
dict

pyanno4rt.learning_model.evaluation

Model evaluation module.

The module aims to provide methods and classes to evaluate the applied learning models.

Subpackages

pyanno4rt.learning_model.evaluation.metrics

Evaluation metrics module.

The module aims to provide methods and classes to evaluate the applied learning models.

Overview

Table 28: Classes

F1Score	F1 metric computation class.
ModelKPI	Model KPI computation class.
PRScore	Precision-Recall scores computation class.
ROCScore	ROC-AUC scores computation class.

Classes

class pyanno4rt.learning_model.evaluation.metrics.**F1Score**(*model_name*, *true_labels*)

F1 metric computation class.

Parameters

- **model_name** (*string*) – Name of the learning model.
- **true_labels** (*ndarray*) – Ground truth values for the labels to predict.

model_name

See 'Parameters'.

Type
string

true_labels

See 'Parameters'.

Type
ndarray

Overview

Table 29: Methods

compute(predicted_labels)	Compute F1 scores.

Members

compute(*predicted_labels*)

Compute F1 scores.

Parameters

predicted_labels (*tuple*) – Tuple of arrays with the labels predicted by the learning model. The first array holds the training prediction labels, the second holds the out-of-folds prediction labels.

Returns

- **f1_scores** (*dict*) – Dictionary with the F1 scores for different thresholds. The keys are 'Training' and 'Out-of-folds', and for each a series of threshold/F1 value pairs is stored.
- **best_f1** (*dict*) – Location of the best F1 score. The keys are 'Training' and 'Out-of-folds', and for each a single threshold value is stored which refers to the maximum F1 score.

class pyanno4rt.learning_model.evaluation.metrics.**ModelKPI**(*model_name*, *true_labels*)

Model KPI computation class.

Parameters

- **model_name** (*string*) – Name of the learning model.
- **true_labels** (*ndarray*) – Ground truth values for the labels to predict.

model_name

See 'Parameters'.

Type

string

true_labels

See 'Parameters'.

Type

ndarray

Overview

Table 30: Methods

compute(predicted_labels, thresholds)	Compute the KPIs.

Members

compute(*predicted_labels*, *thresholds=(0.5, 0.5)*)

Compute the KPIs.

Parameters

- **predicted_labels** (*tuple*) – Tuple of arrays with the labels predicted by the learning model. The first array holds the training prediction labels, the second holds the out-of-folds prediction labels.
- **thresholds** (*tuple, default=(0.5, 0.5)*) – Probability thresholds for the binarization of the probability predictions.

Returns

indicators – Dictionary with the key performance indicators. The keys are 'Training' and 'Out-of-folds', and for each a dictionary with indicator/value pairs is stored.

Return type

dict

class pyanno4rt.learning_model.evaluation.metrics.**PRScore**(*model_name*, *true_labels*)

Precision-Recall scores computation class.

Parameters

- **model_name** (*string*) – Name of the learning model.
- **true_labels** (*ndarray*) – Ground truth values for the labels to predict.

model_name

See 'Parameters'.

Type

string

true_labels

See 'Parameters'.

Type

ndarray

Overview

Table 31: Methods

compute(predicted_labels)	Compute the precision and the recall (curves).

Members

compute(*predicted_labels*)

Compute the precision and the recall (curves).

Parameters

predicted_labels (*tuple*) – Tuple of arrays with the labels predicted by the learning model. The first array holds the training prediction labels, the second holds the out-of-folds prediction labels.

Returns
precision_recall – Dictionary with the precision-recall scores. The keys are 'Training' and 'Out-of-folds', and for each a dataframe with precision-recall scores is stored.

Return type
dict

class pyanno4rt.learning_model.evaluation.metrics.**ROCScore**(*model_name*, *true_labels*)

ROC-AUC scores computation class.

Parameters

- **model_name** (*string*) – Name of the learning model.
- **true_labels** (*ndarray*) – Ground truth values for the labels to predict.

model_name

See 'Parameters'.

Type
string

true_labels

See 'Parameters'.

Type
ndarray

Overview

Table 32: Methods

compute(predicted_labels)	Compute the ROC-AUC (curve).

Members

compute(*predicted_labels*)

Compute the ROC-AUC (curve).

Parameters
predicted_labels (*tuple*) – Tuple of arrays with the labels predicted by the learning model. The first array holds the training prediction labels, the second holds the out-of-folds prediction labels.

Returns

- **scores** (*dict*) – Dictionary with the ROC-AUC scores. The keys are 'Training' and 'Out-of-folds', and for each a dataframe with false positive rates, true positive rates, and thresholds is stored.
- **auc_value** (*dict*) – Dictionary with the AUC values. The keys are 'Training' and 'Out-of-folds', and for each a single AUC value is stored.

Overview

Table 33: Classes

ModelEvaluator	Model evaluation class.

Classes

class pyanno4rt.learning_model.evaluation.**ModelEvaluator**(*model_name*, *true_labels*)

Model evaluation class.

This class provides a collection of evaluation metrics to be computed in a single method call.

Parameters

- **model_name** (*string*) – Name of the learning model.
- **true_labels** (*ndarray*) – Ground truth values for the labels to predict.

model_name

See 'Parameters'.

Type
string

true_labels

See 'Parameters'.

Type
ndarray

evaluations

Dictionary with the evaluation metrics.

Type
dict

Overview

Table 34: Methods

compute(predicted_labels)	Compute the evaluation metrics.

Members

compute(*predicted_labels*)

Compute the evaluation metrics.

Parameters
predicted_labels (*tuple*) – Tuple of arrays with the labels predicted by the learning model. The first array holds the training prediction labels, the second holds the out-of-folds prediction labels.

pyanno4rt.learning_model.features

Features module.

The module aims to provide methods and classes to handle the features of the base data set, i.e., mapping features to segments and definitions from the feature catalogue and iteratively (re)calculate the values as input to the learning model. In addition, the module contains the feature catalogue.

Subpackages

pyanno4rt.learning_model.features.catalogue

Feature catalogue module.

The module aims to provide methods and classes to compute and differentiate dosiomic, radiomic and demographic features. It is designed to be an extensible catalogue which holds all available feature definitions.

Overview

Table 35: Classes

DosiomicFeature	Abstract superclass for dosiomic features.
RadiomicFeature	Abstract superclass for radiomic features.
DemographicFeature	Abstract superclass for demographic features.
DoseMean	Dose mean feature class.
DoseDeviation	Dose deviation feature class.
DoseMaximum	Dose maximum feature class.
DoseMinimum	Dose minimum feature class.
DoseSkewness	Dose skewness feature class.
DoseKurtosis	Dose kurtosis feature class.
DoseEntropy	Dose entropy feature class.
DoseEnergy	Dose energy feature class.
DoseNVoxels	Dose voxel number feature class.
DoseDx	Dose-volume histogram abscissa feature class.
DoseVx	Dose-volume histogram ordinate feature class.
DoseSubvolume	Subvolume dose feature class.
DoseGradient	Dose gradient feature class.
DoseMoment	Dose moment feature class.
SegmentArea	Segment area feature class.
SegmentVolume	Segment volume feature class.
SegmentEigenvalues	Segment eigenvalues feature class.
SegmentEccentricity	Segment eccentricity feature class.
SegmentDensity	Segment density feature class.
SegmentSphericity	Segment sphericity feature class.
SegmentEigenmin	Segment minimum eigenvalue feature class.
SegmentEigenmid	Segment middle eigenvalue feature class.
SegmentEigenmax	Segment maximum eigenvalue feature class.
PatientAge	Patient age feature class.
PatientSex	Patient sex feature class.
PatientDaysafterrt	Patient days-after-radiotherapy feature class.

Classes

class pyanno4rt.learning_model.features.catalogue.**DosiomicFeature**

Abstract superclass for dosiomic features.

Overview

Table 36: Attributes

feature_class	-
value_function	-
gradient_function	-
value_is_jitted	-
gradient_is_jitted	-

Table 37: Methods

compute(dose, *args)	abc Abstract method for computing the feature value.
differentiate(dose, *args)	abc Abstract method for differentiating the feature.

Members

feature_class = 'Dosiomics'

value_function

gradient_function

value_is_jitted = False

gradient_is_jitted = False

abstract compute(*dose*, **args*)

Abstract method for computing the feature value.

abstract differentiate(*dose*, **args*)

Abstract method for differentiating the feature.

class pyanno4rt.learning_model.features.catalogue.**RadiomicFeature**

Abstract superclass for radiomic features.

Overview

Table 38: Attributes

feature_class	-

Table 39: Methods

compute(mask, spacing)	abc Abstract method for computing the feature value.

Members

feature_class = 'Radiomics'

abstract compute(*mask*, *spacing*)

Abstract method for computing the feature value.

class pyanno4rt.learning_model.features.catalogue.**DemographicFeature**

Abstract superclass for demographic features.

Overview

Table 40: Attributes

feature_class	-

Table 41: Methods

compute(value)	abc Abstract method for computing the feature value.

Members

feature_class = 'Demographics'

abstract compute(*value*)

Abstract method for computing the feature value.

class pyanno4rt.learning_model.features.catalogue.**DoseMean**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose mean feature class.

Overview

Table 42: Methods

function(dose)	static Compute the mean dose.
compute(dose, *args)	static Check the jitting status and call the computation function.
differentiate(dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*dose*)

Compute the mean dose.

static compute(*dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseDeviation**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose deviation feature class.

Overview

Table 43: Methods

function(dose)	static Compute the standard deviation of the dose.
compute(dose, *args)	static Check the jitting status and call the computation function.
differentiate(dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*dose*)

Compute the standard deviation of the dose.

static compute(*dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseMaximum**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose maximum feature class.

Overview

Table 44: Methods

function(dose)	static Compute the maximum dose.
compute(dose, *args)	static Check the jitting status and call the computation function.
differentiate(dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*dose*)

Compute the maximum dose.

static compute(*dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseMinimum**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose minimum feature class.

Overview

Table 45: Methods

function(dose)	static Compute the minimum dose.
compute(dose, *args)	static Check the jitting status and call the computation function.
differentiate(dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*dose*)

Compute the minimum dose.

static compute(*dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseSkewness**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose skewness feature class.

Overview

Table 46: Methods

function(dose)	static Compute the skewness.
compute(dose, *args)	static Check the jitting status and call the computation function.
differentiate(dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*dose*)

Compute the skewness.

static compute(*dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseKurtosis**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose kurtosis feature class.

Overview

Table 47: Methods

function(dose)	static Compute the kurtosis.
compute(dose, *args)	static Check the jitting status and call the computation function.
differentiate(dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*dose*)

Compute the kurtosis.

static compute(*dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseEntropy**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose entropy feature class.

Overview

Table 48: Methods

function(dose)	static Compute the entropy.
gradient(dose)	static Compute the entropy gradient.
compute(dose, *args)	static Call the computation function.
differentiate(dose, *args)	static Call the differentiation function.

Members

static function(*dose*)

Compute the entropy.

static gradient(*dose*)

Compute the entropy gradient.

static compute(*dose*, **args*)

Call the computation function.

static differentiate(*dose*, **args*)

Call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseEnergy**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose energy feature class.

Overview

Table 49: Methods

function(dose)	static Compute the energy.
gradient(dose)	static Compute the energy gradient.
compute(dose, *args)	static Call the computation function.
differentiate(dose, *args)	static Call the differentiation function.

Members

static function(*dose*)

Compute the energy.

static gradient(*dose*)

Compute the energy gradient.

static compute(*dose*, **args*)

Call the computation function.

static differentiate(*dose*, **args*)

Call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseNVoxels**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose voxel number feature class.

Overview

Table 50: Methods

function(dose)	static Compute the number of voxels.
compute(dose, *args)	static Check the jitting status and call the computation function.
differentiate(dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*dose*)

Compute the number of voxels.

static compute(*dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseDx**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose-volume histogram abscissa feature class.

Overview

Table 51: Methods

pyfunction(level, dose)	static Compute the dose-volume histogram abscissa in 'python' mode.
matfunction(level, dose)	static Compute the dose-volume histogram abscissa in 'matlab' mode.
compute(level, dose, *args)	static Check the jitting status and call the computation function.
differentiate(level, dose, *args)	static Check the jitting status and call the differentiation function.

Members

static pyfunction(*level*, *dose*)

Compute the dose-volume histogram abscissa in 'python' mode.

static matfunction(*level*, *dose*)

Compute the dose-volume histogram abscissa in 'matlab' mode.

static compute(*level*, *dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*level*, *dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseVx**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose-volume histogram ordinate feature class.

Overview

Table 52: Methods

function(level, dose)	static Compute the dose-volume histogram ordinate.
compute(level, dose, *args)	static Check the jitting status and call the computation function.
differentiate(level, dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*level*, *dose*)

Compute the dose-volume histogram ordinate.

static compute(*level*, *dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*level*, *dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseSubvolume**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Subvolume dose feature class.

Overview

Table 53: Methods

function(subvolume, _, *args)	static Compute the subvolume dose.
compute(subvolume, dose, *args)	static Check the jitting status and call the computation function.
differentiate(subvolume, dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*subvolume*, _, **args*)

Compute the subvolume dose.

static compute(*subvolume*, *dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*subvolume*, *dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseGradient**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose gradient feature class.

Overview

Table 54: Methods

function(axis, dose, *args)	static Compute the dose gradient.
compute(axis, dose, *args)	static Check the jitting status and call the computation function.
differentiate(axis, dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*axis*, *dose*, **args*)

Compute the dose gradient.

static compute(*axis*, *dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*axis*, *dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**DoseMoment**

Bases: *pyanno4rt.learning_model.features.catalogue.DosiomicFeature*

Dose moment feature class.

Overview

Table 55: Methods

function(coefficients, _, *args)	static Compute the dose moment.
compute(coefficients, dose, *args)	static Check the jitting status and call the computation function.
differentiate(coefficients, dose, *args)	static Check the jitting status and call the differentiation function.

Members

static function(*coefficients*, _, **args*)

Compute the dose moment.

static compute(*coefficients*, *dose*, **args*)

Check the jitting status and call the computation function.

static differentiate(*coefficients*, *dose*, **args*)

Check the jitting status and call the differentiation function.

class pyanno4rt.learning_model.features.catalogue.**SegmentArea**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment area feature class.

Overview

Table 56: Methods

compute(mask, spacing)	static Compute the area.

Members

static compute(*mask*, *spacing*)

Compute the area.

class pyanno4rt.learning_model.features.catalogue.**SegmentVolume**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment volume feature class.

Overview

Table 57: Methods

compute(mask, spacing)	static Compute the volume.

Members

static compute(*mask*, *spacing*)

Compute the volume.

class pyanno4rt.learning_model.features.catalogue.**SegmentEigenvalues**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment eigenvalues feature class.

Overview

Table 58: Methods

compute(mask, spacing)	static Compute all eigenvalues.

Members

static compute(*mask*, *spacing*)

Compute all eigenvalues.

class pyanno4rt.learning_model.features.catalogue.**SegmentEccentricity**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment eccentricity feature class.

Overview

Table 59: Methods

compute(mask, spacing)	static Compute the eccentricity.

Members

static compute(*mask*, *spacing*)

Compute the eccentricity.

class pyanno4rt.learning_model.features.catalogue.**SegmentDensity**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment density feature class.

Overview

Table 60: Methods

compute(mask, spacing)	static Compute the density.

Members

static compute(*mask*, *spacing*)

Compute the density.

class pyanno4rt.learning_model.features.catalogue.**SegmentSphericity**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment sphericity feature class.

Overview

Table 61: Methods

compute(mask, spacing)	static Compute the sphericity.

Members

static compute(*mask*, *spacing*)

Compute the sphericity.

class pyanno4rt.learning_model.features.catalogue.**SegmentEigenmin**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment minimum eigenvalue feature class.

Overview

Table 62: Methods

compute(mask, spacing)	static Compute the minimum eigenvalue.

Members

static compute(*mask*, *spacing*)

Compute the minimum eigenvalue.

class pyanno4rt.learning_model.features.catalogue.**SegmentEigenmid**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment middle eigenvalue feature class.

Overview

Table 63: Methods

compute(mask, spacing)	static Compute the middle eigenvalue.

Members

static compute(*mask*, *spacing*)

Compute the middle eigenvalue.

class pyanno4rt.learning_model.features.catalogue.**SegmentEigenmax**

Bases: *pyanno4rt.learning_model.features.catalogue.RadiomicFeature*

Segment maximum eigenvalue feature class.

Overview

Table 64: Methods

compute(mask, spacing)	static Compute the maximum eigenvalue.

Members

static compute(*mask*, *spacing*)

Compute the maximum eigenvalue.

class pyanno4rt.learning_model.features.catalogue.**PatientAge**

Bases: *pyanno4rt.learning_model.features.catalogue.DemographicFeature*

Patient age feature class.

Overview

Table 65: Methods

compute(value)	static Get the age.

Members

static compute(*value*)

Get the age.

class pyanno4rt.learning_model.features.catalogue.**PatientSex**

Bases: *pyanno4rt.learning_model.features.catalogue.DemographicFeature*

Patient sex feature class.

Overview

Table 66: Methods

compute(value)	static Get the sex.

Members

static compute(*value*)

Get the sex.

class pyanno4rt.learning_model.features.catalogue.**PatientDaysafterrt**

Bases: *pyanno4rt.learning_model.features.catalogue.DemographicFeature*

Patient days-after-radiotherapy feature class.

Overview

Table 67: Methods

compute(value)	static Get the days-after-radiotherapy.

Members

static compute(*value*)

Get the days-after-radiotherapy.

Overview

Table 68: Classes

FeatureMapGenerator	Feature map generation class.
FeatureCalculator	Feature value and gradient (re)calculation class.

Classes

class pyanno4rt.learning_model.features.**FeatureMapGenerator**(*model_label*, *fuzzy_matching*)

Feature map generation class.

This class provides a mapping between the features from the data set, the structures from the segmentation, and the definitions from the feature catalogue. Matching is based on fuzzy or exact string matching.

Parameters

fuzzy_matching (*bool*) – Indicator for the use of fuzzy string matching (if 'False', exact string matching is applied).

fuzzy_matching

See 'Parameters'.

Type

bool

feature_map

Dictionary with information on the mapping of features in the dataset with the segmented structures and their computation/differentiation functions.

Type
dict

Notes

In the current implementation, string matching works best if:

- names from segments in the segmentation do not have any special characters except "_" (which will automatically be removed before matching);
- feature names follow the scheme *<name of the segment> _<name of the feature in the catalogue>_<optional parameters>*, e.g. "parotidLeft_doseMean" (mean dose to the left parotid) or "parotidRight_doseGradient_x" (dose gradient in x-direction for the right parotid).

Overview

Table 69: Methods

generate(data_information)	Generate the feature map by fuzzy or exact string matching.

Members

generate(*data_information*)

Generate the feature map by fuzzy or exact string matching.

Parameters
...

Returns
feature_map – Dictionary with information on the mapping of features in the dataset with the segmented structures and their computation/differentiation functions.

Return type
dict

class pyanno4rt.learning_model.features.**FeatureCalculator**(*write_features*, *verbose=True*)

Feature value and gradient (re)calculation class.

Parameters
write_features (*bool*) – Indicator for tracking the feature values.

write_features

See 'Parameters'.

Type
bool

feature_history

Feature values per iteration. If `write_features` is False, this attribute is not set.

Type
ndarray or None

gradient_history

Gradient matrices per iteration. If `write_gradients` is False, this attribute is not set.

Type
list or None

radiomics

Dictionary for mapping the radiomic feature names to the radiomic feature values. It allows to retrieve the feature values after first computation and thus prevents unnecessary recalculation.

Type
dict

demographics

Dictionary for mapping the demographic feature names to the demographic feature values. It allows to retrieve the feature values after first computation and thus prevents unnecessary recalculation.

Type
dict

feature_inputs

Dictionary for collecting the candidate feature input values. This allows to centralize the input retrieval for all calculations.

Type
dict

__iteration__

Iteration numbers for the feature calculation and the optimization problem. By keeping the two elements the same, it is assured that the feature calculator is only active for new problem iterations, rather than per evaluation step.

Type
list

__dose_cache__

Cache array for the dose values.

Type
ndarray

__feature_cache__

Cache array for the feature values.

Type
ndarray

Overview

Table 70: Methods

add_feature_map(feature_map, return_self)	Add the feature map to the calculator.
precompute(dose, segment)	Precompute the dose, dose cube and segment masks as inputs for the feature calculation.
featurize(dose, segment, no_cache)	Convert dose and segment information into the feature vector.
get_feature_vector()	Get the feature vector.
gradientize(dose, segment)	Convert dose and segment information into the gradient matrix.

Members

add_feature_map(*feature_map*, *return_self=False*)

Add the feature map to the calculator.

Parameters

feature_map (*dict*) – ...

precompute(*dose*, *segment*)

Precompute the dose, dose cube and segment masks as inputs for the feature calculation.

Parameters

- **dose** (*tuple of ndarray*) – Value of the dose for a single or multiple segments.
- **segment** (*list of strings*) – Names of the segments associated with the dose.

featurize(*dose*, *segment*, *no_cache=False*)

Convert dose and segment information into the feature vector.

Parameters

- **dose** (*tuple of ndarray*) – Value of the dose for a single or multiple segments.
- **segment** (*list of strings*) – Names of the segments associated with the dose.

Returns

Values of the calculated features.

Return type

ndarray

get_feature_vector()

Get the feature vector.

Parameters

- **dose** (*tuple of ndarray*) – Value of the dose for a single or multiple segments.
- **segment** (*list of strings*) – Names of the segments associated with the dose.

Returns

Values of the calculated features.

Return type

ndarray

gradientize(*dose*, *segment*)

Convert dose and segment information into the gradient matrix.

Parameters

- **dose** (*tuple of ndarray*) – Value of the dose for a single or multiple segments.
- **segment** (*list of strings*) – Names of the segments associated with the dose.

Returns
Matrix of the calculated gradients.

Return type
csr_matrix

pyanno4rt.learning_model.frequentist

Frequentist learning models module.

The module aims to provide methods and classes for modeling NTCP and TCP with frequentist learning models, e.g. logistic regression, neural networks and support vector machines, including individual preprocessing and evaluation pipelines and Bayesian hyperparameter optimization with k-fold cross-validation.

Subpackages

pyanno4rt.learning_model.frequentist.additional_files

Additional model files module.

The module aims to provide functions as a supplement for the frequentist learning models.

Overview

Table 71: Function

build_iocnn(input_shape, output_shape, labels, hyperparameters, squash_output)	Build the input-output convex neural network architecture with the functional API.
build_standard_nn(input_shape, output_shape, labels, hyperparameters, squash_output)	Build the standard neural network architecture with the functional API.
linear_decision_function(svm, features)	Compute the linear decision function for the SVM.
rbf_decision_function(svm, features)	Compute the rbf decision function for the SVM.
poly_decision_function(svm, features)	Compute the poly decision function for the SVM.
sigmoid_decision_function(svm, features)	Compute the sigmoid decision function for the SVM.
linear_decision_gradient(svm, _)	Compute the linear decision function gradient for the SVM.
rbf_decision_gradient(svm, features)	Compute the rbf decision function gradient for the SVM.
poly_decision_gradient(svm, features)	Compute the poly decision function gradient for the SVM.
sigmoid_decision_gradient(svm, features)	Compute the sigmoid decision function gradient for the SVM.

Table 72: Attributes

loss_map	-
optimizer_map	-

Functions

`pyanno4rt.learning_model.frequentist.additional_files.`**`build_iocnn`**(*input_shape*, *output_shape*, *labels*, *hyperparameters*, *squash_output*)

Build the input-output convex neural network architecture with the functional API.

Parameters

- **input_shape** (*int*) – Shape of the input features.
- **output_shape** (*int*) – Shape of the output labels.
- **hyperparameters** (*dict*) – Dictionary with the hyperparameter names and values for the neural network outcome prediction model.
- **squash_output** (*bool*) – Indicator for the use of a sigmoid activation function in the output layer.

Returns

Instance of the class *Functional*, which provides a functional input-output convex neural network architecture.

Return type

object of class 'Functional'

`pyanno4rt.learning_model.frequentist.additional_files.`**`build_standard_nn`**(*input_shape*, *output_shape*, *labels*, *hyperparameters*, *squash_output*)

Build the standard neural network architecture with the functional API.

Parameters

- **input_shape** (*int*) – Shape of the input features.
- **output_shape** (*int*) – Shape of the output labels.
- **hyperparameters** (*dict*) – Dictionary with the hyperparameter names and values for the neural network outcome prediction model.
- **squash_output** (*bool*) – Indicator for the use of a sigmoid activation function in the output layer.

Returns

Instance of the class *Functional*, which provides a functional standard neural network architecture.

Return type

object of class 'Functional'

`pyanno4rt.learning_model.frequentist.additional_files.`**`linear_decision_function`**(*svm*, *features*)

Compute the linear decision function for the SVM.

Parameters

- **svm** (object of class *SVC*) – Instance of scikit-learn's *SVC* class.
- **features** (*ndarray*) – Vector of feature values.

Returns
Value of the decision function with linear kernel.

Return type
float

`pyanno4rt.learning_model.frequentist.additional_files.`**`rbf_decision_function`**(*svm*, *features*)

Compute the rbf decision function for the SVM.

Parameters

- **svm** (object of class *SVC*) – Instance of scikit-learn's *SVC* class.
- **features** (*ndarray*) – Vector of feature values.

Returns
Value of the decision function with rbf kernel.

Return type
float

`pyanno4rt.learning_model.frequentist.additional_files.`**`poly_decision_function`**(*svm*, *features*)

Compute the poly decision function for the SVM.

Parameters

- **svm** (object of class *SVC*) – Instance of scikit-learn's *SVC* class.
- **features** (*ndarray*) – Vector of feature values.

Returns
Value of the decision function with poly kernel.

Return type
float

`pyanno4rt.learning_model.frequentist.additional_files.`**`sigmoid_decision_function`**(*svm*, *features*)

Compute the sigmoid decision function for the SVM.

Parameters

- **svm** (object of class *SVC*) – Instance of scikit-learn's *SVC* class.
- **features** (*ndarray*) – Vector of feature values.

Returns
Value of the decision function with sigmoid kernel.

Return type
float

`pyanno4rt.learning_model.frequentist.additional_files.`**`linear_decision_gradient`**(*svm*, _)

Compute the linear decision function gradient for the SVM.

Parameters
svm (object of class *SVC*) – Instance of scikit-learn's *SVC* class.

Returns
Gradient of the decision function with linear kernel.

Return type
ndarray

`pyanno4rt.learning_model.frequentist.additional_files.`**`rbf_decision_gradient`**(*svm*, *features*)

Compute the rbf decision function gradient for the SVM.

Parameters

- **svm** (object of class *SVC*) – Instance of scikit-learn's *SVC* class.
- **features** (*ndarray*) – Vector of feature values.

Returns

Gradient of the decision function with rbf kernel.

Return type

ndarray

`pyanno4rt.learning_model.frequentist.additional_files.`**`poly_decision_gradient`**(*svm*, *features*)

Compute the poly decision function gradient for the SVM.

Parameters

- **svm** (object of class *SVC*) – Instance of scikit-learn's *SVC* class.
- **features** (*ndarray*) – Vector of feature values.

Returns

Gradient of the decision function with poly kernel.

Return type

ndarray

`pyanno4rt.learning_model.frequentist.additional_files.`**`sigmoid_decision_gradient`**(*svm*, *features*)

Compute the sigmoid decision function gradient for the SVM.

Parameters

- **svm** (object of class *SVC*) – Instance of scikit-learn's *SVC* class.
- **features** (*ndarray*) – Vector of feature values.

Returns

Gradient of the decision function with sigmoid kernel.

Return type

ndarray

Attributes

`pyanno4rt.learning_model.frequentist.additional_files.`**`loss_map`**

`pyanno4rt.learning_model.frequentist.additional_files.`**`optimizer_map`**

Overview

Table 73: Classes

DecisionTreeModel	Decision tree outcome prediction model class.
KNeighborsModel	K-nearest neighbors outcome prediction model class.
LogisticRegressionModel	Logistic regression outcome prediction model class.
NaiveBayesModel	Naive Bayes outcome prediction model class.
NeuralNetworkModel	Neural network outcome prediction model class.
RandomForestModel	Random forest outcome prediction model class.
SupportVectorMachineModel	Support vector machine outcome prediction model class.

Classes

class pyanno4rt.learning_model.frequentist.**DecisionTreeModel**(*model_label*, *model_folder_path*, *dataset*, *preprocessing_steps*, *tune_space*, *tune_evaluations*, *tune_score*, *tune_splits*, *inspect_model*, *evaluate_model*, *oof_splits*, *display_options*)

Decision tree outcome prediction model class.

This class enables building an individual preprocessing pipeline, fit the decision tree model from the input data, inspect the model, make predictions with the model, and assess the predictive performance using multiple evaluation metrics.

The training process includes sequential model-based hyperparameter optimization with tree-structured Parzen estimators and stratified k-fold cross-validation for the objective function evaluation. Cross-validation is also applied to (optionally) inspect the validation feature importances and to generate out-of-folds predictions as a full reconstruction of the input labels for generalization assessment.

Parameters

- **model_label** (*string*) – Label for the decision tree model to be used for file naming.
- **dataset** (*dict*) – Dictionary with the raw data set, the label viewpoint, the label bounds, the feature values and names, and the label values and names after modulation. In a compact way, this represents the input data for the decision tree model.
- **preprocessing_steps** (*tuple*) – Sequence of labels associated with preprocessing algorithms which make up the preprocessing pipeline for the decision tree model. Current available algorithm labels are:
  - transformers : 'Equalizer', 'StandardScaler', 'Whitening'.
- **tune_space** (*dict*) – Search space for the Bayesian hyperparameter optimization, including
  - 'criterion' : measure for the quality of a split;
  - 'splitter' : splitting strategy at each node;
  - 'max_depth' : maximum depth of the tree;
  - 'min_samples_split' : minimum number of samples required for splitting each node;
  - 'min_samples_leaf' : minimum number of samples required at each node;

  - 'min_weight_fraction_leaf' : minimum weighted fraction of the weights sum required at each node;
  - 'max_features' : maximum number of features taken into account when looking for the best split at each node;
  - 'class_weight' : weights associated with the classes;
  - 'ccp_alpha' : complexity parameter for minimal cost-complexity pruning.
- **tune_evaluations** (*int*) – Number of evaluation steps (trials) for the Bayesian hyperparameter optimization.
- **tune_score** (*string*) – Scoring function for the evaluation of the hyperparameter set candidates. Current available scorers are:
  - 'log_loss' : negative log-likelihood score;
  - 'roc_auc_score' : area under the ROC curve score.
- **tune_splits** (*int*) – Number of splits for the stratified cross-validation within each hyperparameter optimization step.
- **inspect_model** (*bool*) – Indicator for the inspection of the model, e.g. the feature importances.
- **evaluate_model** (*bool*) – Indicator for the evaluation of the model, e.g. the model KPIs.
- **oof_splits** (*int*) – Number of splits for the stratified cross-validation within the out-of-folds evaluation step of the decision tree model.

preprocessor

Instance of the class *DataPreprocessor*, which holds methods to build the preprocessing pipeline, fit with the input features, transform the features, and derive the gradient of the preprocessing algorithms w.r.t the features.

Type
object of class *DataPreprocessor*

features

Values of the input features.

Type
ndarray

labels

Values of the input labels.

Type
ndarray

configuration

Dictionary with information for the modeling, i.e., the dataset, the preprocessing, and the hyperparameter search space.

Type
dict

model_path

Path for storing and retrieving the decision tree model.

Type
string

`configuration_path`

Path for storing and retrieving the configuration dictionary.

Type
string

`hyperparameter_path`

Path for storing and retrieving the hyperparameter dictionary.

Type
string

`updated_model`

Indicator for the update status of the model, triggers recalculating the model inspection and model evaluation classes.

Type
bool

`prediction_model`

Instance of the class *DecisionTreeClassifier*, which holds methods to make predictions from the decision tree model.

Type
object of class *DecisionTreeClassifier*

`inspector`

Instance of the class *ModelInspector*, which holds methods to compute model inspection values, e.g. feature importances.

Type
object of class *ModelInspector*

`training_prediction`

Array with the label predictions on the input data.

Type
ndarray

`oof_prediction`

Array with the out-of-folds predictions on the input data.

Type
ndarray

`evaluator`

Instance of the class *ModelEvaluator*, which holds methods to compute the evaluation metrics for a given array with label predictions.

Type
object of class *ModelEvaluator*

Notes

Currently, the preprocessing pipeline for the model is restricted to transformations of the input feature values, e.g. scaling, dimensionality reduction or feature engineering. Transformations which affect the input labels in the same way, e.g. resampling or outlier removal, are not yet possible.

Overview

Table 74: Methods

preprocess(features)	Preprocess the input feature vector with the built pipeline.
get_model(features, labels)	Get the decision tree outcome prediction model by reading from the model file path, the datahub, or by training.
tune_hyperparamet labels)	Tune the hyperparameters of the decision tree model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.
train(features, labels)	Train the decision tree outcome prediction model.
predict(features)	Predict the label values from the feature values.
predict_oof(feature labels, oof_splits)	Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.
inspect(labels)	.
evaluate(features, labels, oof_splits)	.
set_file_paths(bas	Set the paths for model, configuration and hyperparameter files.
read_model_from_f	Read the decision tree outcome prediction model from the model file path.
write_model_to_fi	Write the decision tree outcome prediction model to the model file path.
read_configuratio	Read the configuration dictionary from the configuration file path.
write_configurati	Write the configuration dictionary to the configuration file path.
read_hyperparamet	Read the decision tree outcome prediction model hyperparameters from the hyperparameter file path.
write_hyperparame	Write the hyperparameter dictionary to the hyperparameter file path.

Members

preprocess(*features*)

Preprocess the input feature vector with the built pipeline.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Array of transformed feature values.

Return type
ndarray

get_model(*features*, *labels*)

Get the decision tree outcome prediction model by reading from the model file path, the datahub, or by training.

Returns
Instance of the class *DecisionTreeClassifier*, which holds methods to make predictions from the decision tree model.

Return type
object of class *DecisionTreeClassifier*

`tune_hyperparameters_with_bayes`(*features*, *labels*)

Tune the hyperparameters of the decision tree model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.

Returns
tuned_hyperparameters – Dictionary with the hyperparameter names and values tuned via Bayesian hyperparameter optimization.

Return type
dict

`train`(*features*, *labels*)

Train the decision tree outcome prediction model.

Returns
prediction_model – Instance of the class *DecisionTreeClassifier*, which holds methods to make predictions from the decision tree model.

Return type
object of class *DecisionTreeClassifier*

`predict`(*features*)

Predict the label values from the feature values.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Floating-point label prediction or array of label predictions.

Return type
float or ndarray

`predict_oof`(*features*, *labels*, *oof_splits*)

Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.

Parameters
oof_splits (*int*) – Number of splits for the stratified cross-validation.

Returns
Array with the out-of-folds label predictions.

Return type
ndarray

`inspect`(*labels*)

.

`evaluate`(*features*, *labels*, *oof_splits*)

.

`set_file_paths`(*base_path*)

Set the paths for model, configuration and hyperparameter files.

Parameters
base_path (*string*) – Base path from which to access the model files.

read_model_from_file()

Read the decision tree outcome prediction model from the model file path.

Returns
Instance of the class *DecisionTreeClassifier*, which holds methods to make predictions from the decision tree model.

Return type
object of class *DecisionTreeClassifier*

write_model_to_file(*prediction_model*)

Write the decision tree outcome prediction model to the model file path.

Parameters
prediction_model (object of class *DecisionTreeClassifier*) – Instance of the class *DecisionTreeClassifier*, which holds methods to make predictions from the decision tree model.

read_configuration_from_file()

Read the configuration dictionary from the configuration file path.

Returns
Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

Return type
dict

write_configuration_to_file(*configuration*)

Write the configuration dictionary to the configuration file path.

Parameters
configuration (*dict*) – Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

read_hyperparameters_from_file()

Read the decision tree outcome prediction model hyperparameters from the hyperparameter file path.

Returns
Dictionary with the hyperparameter names and values for the decision tree outcome prediction model.

Return type
dict

write_hyperparameters_to_file(*hyperparameters*)

Write the hyperparameter dictionary to the hyperparameter file path.

Parameters
hyperparameters (*dict*) – Dictionary with the hyperparameter names and values for the decision tree outcome prediction model.

class pyanno4rt.learning_model.frequentist.**KNeighborsModel**(*model_label*, *model_folder_path*, *dataset*, *preprocessing_steps*, *tune_space*, *tune_evaluations*, *tune_score*, *tune_splits*, *inspect_model*, *evaluate_model*, *oof_splits*, *display_options*)

K-nearest neighbors outcome prediction model class.

This class enables building an individual preprocessing pipeline, fit the k-nearest neighbors model from the input data, inspect the model, make predictions with the model, and assess the predictive performance using multiple evaluation metrics.

The training process includes sequential model-based hyperparameter optimization with tree-structured Parzen estimators and stratified k-fold cross-validation for the objective function evaluation. Cross-validation is also applied to (optionally) inspect the validation feature importances and to generate out-of-folds predictions as a full reconstruction of the input labels for generalization assessment.

Parameters

- **model_label** (*string*) – Label for the k-nearest neighbors model to be used for file naming.
- **dataset** (*dict*) – Dictionary with the raw data set, the label viewpoint, the label bounds, the feature values and names, and the label values and names after modulation. In a compact way, this represents the input data for the k-nearest neighbors model.
- **preprocessing_steps** (*tuple*) – Sequence of labels associated with preprocessing algorithms which make up the preprocessing pipeline for the k-nearest neighbors model. Current available algorithm labels are:
  - transformers : 'Equalizer', 'StandardScaler', 'Whitening'.
- **tune_space** (*dict*) – Search space for the Bayesian hyperparameter optimization, including
  - 'n_neighbors' : number of neighbors (equals k);
  - 'weights' : weights function on the neighbors for prediction;
  - 'algorithm' : algorithm for the computation of the neighbors;
  - 'leaf_size' : leaf size for BallTree or KDTree;
  - 'p' : power parameter for the Minkowski metric.
- **tune_evaluations** (*int*) – Number of evaluation steps (trials) for the Bayesian hyperparameter optimization.
- **tune_score** (*string*) – Scoring function for the evaluation of the hyperparameter set candidates. Current available scorers are:
  - 'log_loss' : negative log-likelihood score;
  - 'roc_auc_score' : area under the ROC curve score.
- **tune_splits** (*int*) – Number of splits for the stratified cross-validation within each hyperparameter optimization step.
- **inspect_model** (*bool*) – Indicator for the inspection of the model, e.g. the feature importances.
- **evaluate_model** (*bool*) – Indicator for the evaluation of the model, e.g. the model KPIs.
- **oof_splits** (*int*) – Number of splits for the stratified cross-validation within the out-of-folds evaluation step of the k-nearest neighbors model.

preprocessor

Instance of the class *DataPreprocessor*, which holds methods to build the preprocessing pipeline, fit with the input features, transform the features, and derive the gradient of the preprocessing algorithms w.r.t the features.

Type

object of class *DataPreprocessor*

`features`

Values of the input features.

Type
ndarray

`labels`

Values of the input labels.

Type
ndarray

`configuration`

Dictionary with information for the modeling, i.e., the dataset, the preprocessing, and the hyperparameter search space.

Type
dict

`model_path`

Path for storing and retrieving the k-nearest neighbors model.

Type
string

`configuration_path`

Path for storing and retrieving the configuration dictionary.

Type
string

`hyperparameter_path`

Path for storing and retrieving the hyperparameter dictionary.

Type
string

`updated_model`

Indicator for the update status of the model, triggers recalculating the model inspection and model evaluation classes.

Type
bool

`prediction_model`

Instance of the class *KNeighborsClassifier*, which holds methods to make predictions from the k-nearest neighbors model.

Type
object of class *KNeighborsClassifier*

`inspector`

Instance of the class *ModelInspector*, which holds methods to compute model inspection values, e.g. feature importances.

Type
object of class *ModelInspector*

training_prediction

Array with the label predictions on the input data.

Type
ndarray

oof_prediction

Array with the out-of-folds predictions on the input data.

Type
ndarray

evaluator

Instance of the class *ModelEvaluator*, which holds methods to compute the evaluation metrics for a given array with label predictions.

Type
object of class *ModelEvaluator*

Notes

Currently, the preprocessing pipeline for the model is restricted to transformations of the input feature values, e.g. scaling, dimensionality reduction or feature engineering. Transformations which affect the input labels in the same way, e.g. resampling or outlier removal, are not yet possible.

Overview

Table 75: Methods

preprocess(features	Preprocess the input feature vector with the built pipeline.
get_model(features, labels)	Get the k-nearest neighbors outcome prediction model by reading from the model file path, the datahub, or by training.
tune_hyperparamet labels)	Tune the hyperparameters of the k-nearest neighbors model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.
train(features, labels)	Train the k-nearest neighbors outcome prediction model.
predict(features)	Predict the label values from the feature values.
predict_oof(feature labels, oof_splits)	Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.
inspect(labels)	.
evaluate(features, labels, oof_splits)	.
set_file_paths(ba	Set the paths for model, configuration and hyperparameter files.
read_model_from_f	Read the k-nearest neighbors outcome prediction model from the model file path.
write_model_to_fi	Write the k-nearest neighbors outcome prediction model to the model file path.
read_configuratic	Read the configuration dictionary from the configuration file path.
write_configurati	Write the configuration dictionary to the configuration file path.
read_hyperparamet	Read the k-nearest neighbors outcome prediction model hyperparameters from the hyperparameter file path.
write_hyperparame	Write the hyperparameter dictionary to the hyperparameter file path.

Members

preprocess(*features*)

Preprocess the input feature vector with the built pipeline.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Array of transformed feature values.

Return type
ndarray

get_model(*features*, *labels*)

Get the k-nearest neighbors outcome prediction model by reading from the model file path, the datahub, or by training.

Returns
Instance of the class *KNeighborsClassifier*, which holds methods to make predictions from the k-nearest neighbors model.

Return type
object of class *KNeighborsClassifier*

tune_hyperparameters_with_bayes(*features*, *labels*)

Tune the hyperparameters of the k-nearest neighbors model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.

Returns
tuned_hyperparameters – Dictionary with the hyperparameter names and values tuned via Bayesian hyperparameter optimization.

Return type
dict

train(*features*, *labels*)

Train the k-nearest neighbors outcome prediction model.

Returns
prediction_model – Instance of the class *KNeighborsClassifier*, which holds methods to make predictions from the k-nearest neighbors model.

Return type
object of class *KNeighborsClassifier*

predict(*features*)

Predict the label values from the feature values.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Floating-point label prediction or array of label predictions.

Return type
float or ndarray

predict_oof(*features*, *labels*, *oof_splits*)

Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.

Parameters
oof_splits (*int*) – Number of splits for the stratified cross-validation.

Returns
Array with the out-of-folds label predictions.

Return type
ndarray

inspect(*labels*)

.

evaluate(*features*, *labels*, *oof_splits*)

.

set_file_paths(*base_path*)

Set the paths for model, configuration and hyperparameter files.

Parameters
base_path (*string*) – Base path from which to access the model files.

read_model_from_file()

Read the k-nearest neighbors outcome prediction model from the model file path.

Returns
Instance of the class *KNeighborsClassifier*, which holds methods to make predictions from the k-nearest neighbors model.

Return type
object of class *KNeighborsClassifier*

write_model_to_file(*prediction_model*)

Write the k-nearest neighbors outcome prediction model to the model file path.

Parameters
prediction_model (object of class *KNeighborsClassifier*) – Instance of the class *KNeighborsClassifier*, which holds methods to make predictions from the k-nearest neighbors model.

read_configuration_from_file()

Read the configuration dictionary from the configuration file path.

Returns
Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

Return type
dict

write_configuration_to_file(*configuration*)

Write the configuration dictionary to the configuration file path.

Parameters
configuration (*dict*) – Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

read_hyperparameters_from_file()

Read the k-nearest neighbors outcome prediction model hyperparameters from the hyperparameter file path.

Returns
Dictionary with the hyperparameter names and values for the k-nearest neighbors outcome prediction model.

Return type
dict

write_hyperparameters_to_file(*hyperparameters*)

Write the hyperparameter dictionary to the hyperparameter file path.

Parameters
hyperparameters (*dict*) – Dictionary with the hyperparameter names and values for the k-nearest neighbors outcome prediction model.

class pyanno4rt.learning_model.frequentist.**LogisticRegressionModel**(*model_label*, *model_folder_path*, *dataset*, *preprocessing_steps*, *tune_space*, *tune_evaluations*, *tune_score*, *tune_splits*, *inspect_model*, *evaluate_model*, *oof_splits*, *display_options*)

Logistic regression outcome prediction model class.

This class enables building an individual preprocessing pipeline, fit the logistic regression model from the input data, inspect the model, make predictions with the model, and assess the predictive performance using multiple evaluation metrics.

The training process includes sequential model-based hyperparameter optimization with tree-structured Parzen estimators and stratified k-fold cross-validation for the objective function evaluation. Cross-validation is also applied to (optionally) inspect the validation feature importances and to generate out-of-folds predictions as a full reconstruction of the input labels for generalization assessment.

Parameters

- **model_label** (*string*) – Label for the logistic regression model to be used for file naming.
- **dataset** (*dict*) – Dictionary with the raw data set, the label viewpoint, the label bounds, the feature values and names, and the label values and names after modulation. In a compact way, this represents the input data for the logistic regression model.
- **preprocessing_steps** (*tuple*) – Sequence of labels associated with preprocessing algorithms which make up the preprocessing pipeline for the logistic regression model. Current available algorithm labels are:
  - transformers : 'Equalizer', 'StandardScaler', 'Whitening'.
- **tune_space** (*dict*) – Search space for the Bayesian hyperparameter optimization, including
  - 'C' : inverse of the regularization strength;
  - 'penalty' : norm of the penalty function;
  - 'tol' : tolerance for stopping criteria;

  - 'class_weight' : weights associated with the classes.
- **tune_evaluations** (*int*) – Number of evaluation steps (trials) for the Bayesian hyperparameter optimization.
- **tune_score** (*string*) – Scoring function for the evaluation of the hyperparameter set candidates. Current available scorers are:
  - 'log_loss' : negative log-likelihood score;
  - 'roc_auc_score' : area under the ROC curve score.
- **tune_splits** (*int*) – Number of splits for the stratified cross-validation within each hyperparameter optimization step.
- **inspect_model** (*bool*) – Indicator for the inspection of the model, e.g. the feature importances.
- **evaluate_model** (*bool*) – Indicator for the evaluation of the model, e.g. the model KPIs.
- **oof_splits** (*int*) – Number of splits for the stratified cross-validation within the out-of-folds evaluation step of the logistic regression model.

preprocessor

Instance of the class *DataPreprocessor*, which holds methods to build the preprocessing pipeline, fit with the input features, transform the features, and derive the gradient of the preprocessing algorithms w.r.t the features.

Type
object of class *DataPreprocessor*

features

Values of the input features.

Type
ndarray

labels

Values of the input labels.

Type
ndarray

configuration

Dictionary with information for the modeling, i.e., the dataset, the preprocessing, and the hyperparameter search space.

Type
dict

model_path

Path for storing and retrieving the logistic regression model.

Type
string

configuration_path

Path for storing and retrieving the configuration dictionary.

Type
string

hyperparameter_path

Path for storing and retrieving the hyperparameter dictionary.

Type
string

updated_model

Indicator for the update status of the model, triggers recalculating the model inspection and model evaluation classes.

Type
bool

prediction_model

Instance of the class *LogisticRegression*, which holds methods to make predictions from the logistic regression model.

Type
object of class *LogisticRegression*

inspector

Instance of the class *ModelInspector*, which holds methods to compute model inspection values, e.g. feature importances.

Type
object of class *ModelInspector*

training_prediction

Array with the label predictions on the input data.

Type
ndarray

oof_prediction

Array with the out-of-folds predictions on the input data.

Type
ndarray

evaluator

Instance of the class *ModelEvaluator*, which holds methods to compute the evaluation metrics for a given array with label predictions.

Type
object of class *ModelEvaluator*

Notes

Currently, the preprocessing pipeline for the model is restricted to transformations of the input feature values, e.g. scaling, dimensionality reduction or feature engineering. Transformations which affect the input labels in the same way, e.g. resampling or outlier removal, are not yet possible.

Overview

Table 76: Methods

preprocess(features	Preprocess the input feature vector with the built pipeline.
get_model(features, labels)	Get the logistic regression outcome prediction model by reading from the model file path, the datahub, or by training.
tune_hyperparamet labels)	Tune the hyperparameters of the logistic regression model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.
train(features, labels)	Train the logistic regression outcome prediction model.
predict(features)	Predict the label values from the feature values.
predict_oof(feature labels, oof_splits)	Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.
inspect(labels)	.
evaluate(features, labels, oof_splits)	.
set_file_paths(ba	Set the paths for model, configuration and hyperparameter files.
read_model_from_f	Read the logistic regression outcome prediction model from the model file path.
write_model_to_fi	Write the logistic regression outcome prediction model to the model file path.
read_configuratic	Read the configuration dictionary from the configuration file path.
write_configurati	Write the configuration dictionary to the configuration file path.
read_hyperparamet	Read the logistic regression outcome prediction model hyperparameters from the hyperparameter file path.
write_hyperparame	Write the hyperparameter dictionary to the hyperparameter file path.

Members

preprocess(*features*)

Preprocess the input feature vector with the built pipeline.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Array of transformed feature values.

Return type
ndarray

get_model(*features*, *labels*)

Get the logistic regression outcome prediction model by reading from the model file path, the datahub, or by training.

Returns
Instance of the class *LogisticRegression*, which holds methods to make predictions from the logistic regression model.

Return type
object of class *LogisticRegression*

tune_hyperparameters_with_bayes(*features*, *labels*)

Tune the hyperparameters of the logistic regression model via sequential model-based optimization using

the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.

Returns
tuned_hyperparameters – Dictionary with the hyperparameter names and values tuned via Bayesian hyperparameter optimization.

Return type
dict

`train`(*features*, *labels*)

Train the logistic regression outcome prediction model.

Returns
prediction_model – Instance of the class *LogisticRegression*, which holds methods to make predictions from the logistic regression model.

Return type
object of class *LogisticRegression*

`predict`(*features*)

Predict the label values from the feature values.

Parameters
`features` (*ndarray*) – Array of input feature values.

Returns
Floating-point label prediction or array of label predictions.

Return type
float or ndarray

`predict_oof`(*features*, *labels*, *oof_splits*)

Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.

Parameters
`oof_splits` (*int*) – Number of splits for the stratified cross-validation.

Returns
Array with the out-of-folds label predictions.

Return type
ndarray

`inspect`(*labels*)

.

`evaluate`(*features*, *labels*, *oof_splits*)

.

`set_file_paths`(*base_path*)

Set the paths for model, configuration and hyperparameter files.

Parameters
`base_path` (*string*) – Base path from which to access the model files.

`read_model_from_file`()

Read the logistic regression outcome prediction model from the model file path.

Returns
Instance of the class *LogisticRegression*, which holds methods to make predictions from the logistic regression model.

Return type
object of class *LogisticRegression*

`write_model_to_file`(*prediction_model*)

Write the logistic regression outcome prediction model to the model file path.

Parameters
prediction_model (object of class *LogisticRegression*) – Instance of the class *LogisticRegression*, which holds methods to make predictions from the logistic regression model.

`read_configuration_from_file`()

Read the configuration dictionary from the configuration file path.

Returns
Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

Return type
dict

`write_configuration_to_file`(*configuration*)

Write the configuration dictionary to the configuration file path.

Parameters
configuration (*dict*) – Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

`read_hyperparameters_from_file`()

Read the logistic regression outcome prediction model hyperparameters from the hyperparameter file path.

Returns
Dictionary with the hyperparameter names and values for the logistic regression outcome prediction model.

Return type
dict

`write_hyperparameters_to_file`(*hyperparameters*)

Write the hyperparameter dictionary to the hyperparameter file path.

Parameters
hyperparameters (*dict*) – Dictionary with the hyperparameter names and values for the logistic regression outcome prediction model.

class pyanno4rt.learning_model.frequentist.**NaiveBayesModel**(*model_label*, *model_folder_path*, *dataset*, *preprocessing_steps*, *tune_space*, *tune_evaluations*, *tune_score*, *tune_splits*, *inspect_model*, *evaluate_model*, *oof_splits*, *display_options*)

Naive Bayes outcome prediction model class.

This class enables building an individual preprocessing pipeline, fit the naive Bayes model from the input data, inspect the model, make predictions with the model, and assess the predictive performance using multiple evaluation metrics.

The training process includes sequential model-based hyperparameter optimization with tree-structured Parzen estimators and stratified k-fold cross-validation for the objective function evaluation. Cross-validation is also

applied to (optionally) inspect the validation feature importances and to generate out-of-folds predictions as a full reconstruction of the input labels for generalization assessment.

Parameters

- **model_label** (*string*) – Label for the naive Bayes model to be used for file naming.
- **dataset** (*dict*) – Dictionary with the raw data set, the label viewpoint, the label bounds, the feature values and names, and the label values and names after modulation. In a compact way, this represents the input data for the naive Bayes model.
- **preprocessing_steps** (*tuple*) – Sequence of labels associated with preprocessing algorithms which make up the preprocessing pipeline for the naive Bayes model. Current available algorithm labels are:
  - transformers : 'Equalizer', 'StandardScaler', 'Whitening'.
- **tune_space** (*dict*) – Search space for the Bayesian hyperparameter optimization, including
  - 'priors' : prior probabilities of the classes;
  - 'var_smoothing' : additional variance for calculation stability.
- **tune_evaluations** (*int*) – Number of evaluation steps (trials) for the Bayesian hyperparameter optimization.
- **tune_score** (*string*) – Scoring function for the evaluation of the hyperparameter set candidates. Current available scorers are:
  - 'log_loss' : negative log-likelihood score;
  - 'roc_auc_score' : area under the ROC curve score.
- **tune_splits** (*int*) – Number of splits for the stratified cross-validation within each hyperparameter optimization step.
- **inspect_model** (*bool*) – Indicator for the inspection of the model, e.g. the feature importances.
- **evaluate_model** (*bool*) – Indicator for the evaluation of the model, e.g. the model KPIs.
- **oof_splits** (*int*) – Number of splits for the stratified cross-validation within the out-of-folds evaluation step of the naive Bayes model.

preprocessor

Instance of the class *DataPreprocessor*, which holds methods to build the preprocessing pipeline, fit with the input features, transform the features, and derive the gradient of the preprocessing algorithms w.r.t the features.

Type
object of class *DataPreprocessor*

features

Values of the input features.

Type
ndarray

labels

Values of the input labels.

Type
ndarray

`configuration`

Dictionary with information for the modeling, i.e., the dataset, the preprocessing, and the hyperparameter search space.

Type
dict

`model_path`

Path for storing and retrieving the naive Bayes model.

Type
string

`configuration_path`

Path for storing and retrieving the configuration dictionary.

Type
string

`hyperparameter_path`

Path for storing and retrieving the hyperparameter dictionary.

Type
string

`updated_model`

Indicator for the update status of the model, triggers recalculating the model inspection and model evaluation classes.

Type
bool

`prediction_model`

Instance of the class *GaussianNB*, which holds methods to make predictions from the naive Bayes model.

Type
object of class *GaussianNB*

`inspector`

Instance of the class *ModelInspector*, which holds methods to compute model inspection values, e.g. feature importances.

Type
object of class *ModelInspector*

`training_prediction`

Array with the label predictions on the input data.

Type
ndarray

`oof_prediction`

Array with the out-of-folds predictions on the input data.

Type
ndarray

`evaluator`

Instance of the class *ModelEvaluator*, which holds methods to compute the evaluation metrics for a given array with label predictions.

Type
object of class *ModelEvaluator*

Notes

Currently, the preprocessing pipeline for the model is restricted to transformations of the input feature values, e.g. scaling, dimensionality reduction or feature engineering. Transformations which affect the input labels in the same way, e.g. resampling or outlier removal, are not yet possible.

Overview

Table 77: Methods

preprocess(features)	Preprocess the input feature vector with the built pipeline.
get_model(features, labels)	Get the naive Bayes outcome prediction model by reading from the model file path, the datahub, or by training.
tune_hyperparamet labels)	Tune the hyperparameters of the naive Bayes model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.
train(features, labels)	Train the naive Bayes outcome prediction model.
predict(features)	Predict the label values from the feature values.
predict_oof(feature labels, oof_splits)	Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.
inspect(labels)	.
evaluate(features, labels, oof_splits)	.
set_file_paths(bas	Set the paths for model, configuration and hyperparameter files.
read_model_from_f	Read the naive Bayes outcome prediction model from the model file path.
write_model_to_fi	Write the naive Bayes outcome prediction model to the model file path.
read_configuratio	Read the configuration dictionary from the configuration file path.
write_configurati	Write the configuration dictionary to the configuration file path.
read_hyperparamet	Read the naive Bayes outcome prediction model hyperparameters from the hyperparameter file path.
write_hyperparame	Write the hyperparameter dictionary to the hyperparameter file path.

Members

preprocess(*features*)

Preprocess the input feature vector with the built pipeline.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Array of transformed feature values.

Return type
ndarray

get_model(*features*, *labels*)

Get the naive Bayes outcome prediction model by reading from the model file path, the datahub, or by training.

Returns
Instance of the class *GaussianNB*, which holds methods to make predictions from the naive Bayes model.

Return type
object of class *GaussianNB*

`tune_hyperparameters_with_bayes`(*features*, *labels*)

Tune the hyperparameters of the naive Bayes model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.

Returns
tuned_hyperparameters – Dictionary with the hyperparameter names and values tuned via Bayesian hyperparameter optimization.

Return type
dict

`train`(*features*, *labels*)

Train the naive Bayes outcome prediction model.

Returns
prediction_model – Instance of the class *GaussianNB*, which holds methods to make predictions from the naive Bayes model.

Return type
object of class *GaussianNB*

`predict`(*features*)

Predict the label values from the feature values.

Parameters
`features` (*ndarray*) – Array of input feature values.

Returns
Floating-point label prediction or array of label predictions.

Return type
float or ndarray

`predict_oof`(*features*, *labels*, *oof_splits*)

Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.

Parameters
`oof_splits` (*int*) – Number of splits for the stratified cross-validation.

Returns
Array with the out-of-folds label predictions.

Return type
ndarray

`inspect`(*labels*)

.

`evaluate`(*features*, *labels*, *oof_splits*)

.

`set_file_paths`(*base_path*)

Set the paths for model, configuration and hyperparameter files.

Parameters
base_path (*string*) – Base path from which to access the model files.

read_model_from_file()

Read the naive Bayes outcome prediction model from the model file path.

Returns
Instance of the class *GaussianNB*, which holds methods to make predictions from the naive Bayes model.

Return type
object of class *GaussianNB*

write_model_to_file(*prediction_model*)

Write the naive Bayes outcome prediction model to the model file path.

Parameters
prediction_model (object of class *GaussianNB*) – Instance of the class *GaussianNB*, which holds methods to make predictions from the naive Bayes model.

read_configuration_from_file()

Read the configuration dictionary from the configuration file path.

Returns
Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

Return type
dict

write_configuration_to_file(*configuration*)

Write the configuration dictionary to the configuration file path.

Parameters
configuration (*dict*) – Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

read_hyperparameters_from_file()

Read the naive Bayes outcome prediction model hyperparameters from the hyperparameter file path.

Returns
Dictionary with the hyperparameter names and values for the naive Bayes outcome prediction model.

Return type
dict

write_hyperparameters_to_file(*hyperparameters*)

Write the hyperparameter dictionary to the hyperparameter file path.

Parameters
hyperparameters (*dict*) – Dictionary with the hyperparameter names and values for the naive Bayes outcome prediction model.

class pyanno4rt.learning_model.frequentist.**NeuralNetworkModel**(*model_label*, *model_folder_path*, *dataset*, *preprocessing_steps*, *architecture*, *max_hidden_layers*, *tune_space*, *tune_evaluations*, *tune_score*, *tune_splits*, *inspect_model*, *evaluate_model*, *oof_splits*, *display_options*)

Neural network outcome prediction model class.

This class enables building an individual preprocessing pipeline, fit the neural network model from the input data, inspect the model, make predictions with the model, and assess the predictive performance using multiple evaluation metrics.

The training process includes sequential model-based hyperparameter optimization with tree-structured Parzen estimators and stratified k-fold cross-validation for the objective function evaluation. Cross-validation is also applied to (optionally) inspect the validation feature importances and to generate out-of-folds predictions as a full reconstruction of the input labels for generalization assessment.

Parameters

- **model_label** (*string*) – Label for the neural network model to be used for file naming.
- **dataset** (*dict*) – Dictionary with the raw data set, the label viewpoint, the label bounds, the feature values and names, and the label values and names after modulation. In a compact way, this represents the input data for the neural network model.
- **preprocessing_steps** (*tuple*) – Sequence of labels associated with preprocessing algorithms which make up the preprocessing pipeline for the neural network model. Current available algorithm labels are:
  - transformers : 'Equalizer', 'StandardScaler', 'Whitening'.
- **architecture** (*{'input-convex', 'standard'}*) – Type of architecture for the neural network model. Current available architectures are:
  - 'input-convex' : builds the input-convex network architecture;
  - 'standard' : builds the standard feed-forward network architecture.
- **max_hidden_layers** (*int*) – Maximum number of hidden layers for the neural network model.
- **tune_space** (*dict*) – Search space for the Bayesian hyperparameter optimization, including
  - 'input_neuron_number' : number of neurons for the input layer;
  - 'input_activation' : activation function for the input layer ('elu', 'exponential', 'gelu', 'linear', 'leaky_relu', 'relu', 'softmax', 'softplus', 'swish');
  - 'hidden_neuron_number' : number of neurons for the hidden layer(s);
  - 'hidden_activation' : activation function for the hidden layer(s) ('elu', 'gelu', 'linear', 'leaky_relu', 'relu', 'softmax', 'softplus', 'swish');
  - 'input_dropout_rate' : dropout rate for the input layer;
  - 'hidden_dropout_rate' : dropout rate for the hidden layer(s);
  - 'batch_size' : batch size;
  - 'learning_rate' : learning rate
  - 'optimizer' : algorithm for the optimization of the network ('Adam', 'Ftrl', 'SGD');
  - 'loss' : loss function for the optimization of the network ('BCE', 'FocalBCE', 'KLD').
- **tune_evaluations** (*int*) – Number of evaluation steps (trials) for the Bayesian hyperparameter optimization.

- **tune_score** (*string*) – Scoring function for the evaluation of the hyperparameter set candidates. Current available scorers are:
  - 'log_loss' : negative log-likelihood score;
  - 'roc_auc_score' : area under the ROC curve score.
- **tune_splits** (*int*) – Number of splits for the stratified cross-validation within each hyperparameter optimization step.
- **inspect_model** (*bool*) – Indicator for the inspection of the model, e.g. the feature importances.
- **inspect_model** – Indicator for the inspection of the model, e.g. the feature importances.
- **evaluate_model** (*bool*) – Indicator for the evaluation of the model, e.g. the model KPIs.
- **oof_splits** (*int*) – Number of splits for the stratified cross-validation within the out-of-folds evaluation step of the logistic regression model.

preprocessor

Instance of the class *DataPreprocessor*, which holds methods to build the preprocessing pipeline, fit with the input features, transform the features, and derive the gradient of the preprocessing algorithms w.r.t the features.

Type
object of class *DataPreprocessor*

features

Values of the input features.

Type
ndarray

labels

Values of the input labels.

Type
ndarray

configuration

Dictionary with information for the modeling, i.e., the dataset, the preprocessing, and the hyperparameter search space.

Type
dict

model_path

Path for storing and retrieving the neural network model.

Type
string

configuration_path

Path for storing and retrieving the configuration dictionary.

Type
string

`hyperparameter_path`

Path for storing and retrieving the hyperparameter dictionary.

Type
string

`updated_model`

Indicator for the update status of the model, triggers recalculating the model inspection and model evaluation classes.

Type
bool

`prediction_model`

Instance of the class *Functional*, which holds methods to make predictions from the neural network model.

Type
object of class *Functional*

`optimization_model`

Instance of the class *Functional*, equivalent to `prediction_model`, but skips the sigmoid output activation.

Type
object of class *Functional*

`inspector`

Instance of the class *ModelInspector*, which holds methods to compute model inspection values, e.g. feature importances.

Type
object of class *ModelInspector*

`training_prediction`

Array with the label predictions on the input data.

Type
ndarray

`oof_prediction`

Array with the out-of-folds predictions on the input data.

Type
ndarray

`evaluator`

Instance of the class *ModelEvaluator*, which holds methods to compute the evaluation metrics for a given array with label predictions.

Type
object of class *ModelEvaluator*

Notes

Currently, the preprocessing pipeline for the model is restricted to transformations of the input feature values, e.g. scaling, dimensionality reduction or feature engineering. Transformations which affect the input labels in the same way, e.g. resampling or outlier removal, are not yet possible.

Overview

Table 78: Methods

preprocess(features)	Preprocess the input feature vector with the built pipeline.
get_prediction_model(feat labels)	Get the neural network outcome prediction model by reading from the model file path, the datahub, or by training.
get_optimization_model(f labels)	Get the neural network outcome optimization model.
build_network(input_shape, output_shape, hyperparameters, squash_output)	Build the neural network architecture with the functional API.
compile_and_fit(prediction features, labels, hyperparameters, validation_data)	Compile and fit the neural network outcome prediction model to the input data.
tune_hyperparameters_wi labels)	Tune the hyperparameters of the neural network model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.
train(features, labels)	Train the neural network outcome prediction model.
predict(features, squash_output)	Predict the label values from the feature values.
predict_oof(features, labels, oof_splits)	Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.
inspect(labels)	.
evaluate(features, labels, oof_splits)	.
set_file_paths(base_path)	Set the paths for model, configuration and hyperparameter files.
read_model_from_file()	Read the neural network outcome prediction model from the model file path.
write_model_to_file(predi	Write the neural network outcome prediction model to the model file path.
*read_configuration_from_*	Read the configuration dictionary from the configuration file path.
*write_configuration_to_*	Write the configuration dictionary to the configuration file path.
read_hyperparameters_fr	Read the neural network outcome prediction model hyperparameters from the hyperparameter file path.
write_hyperparameters_t	Write the hyperparameter dictionary to the hyperparameter file path.

Members

preprocess(*features*)

Preprocess the input feature vector with the built pipeline.

Parameters

features (*ndarray*) – Array of input feature values.

Returns

Array of transformed feature values.

Return type
ndarray

`get_prediction_model`(*features*, *labels*)

Get the neural network outcome prediction model by reading from the model file path, the datahub, or by training.

Returns
Instance of the class *Functional*, which holds methods to make predictions from the neural network model.

Return type
object of class *Functional*

`get_optimization_model`(*features*, *labels*)

Get the neural network outcome optimization model.

Returns
Instance of the class *Functional*, which holds methods to make predictions from the neural network model.

Return type
object of class *Functional*

`build_network`(*input_shape*, *output_shape*, *hyperparameters*, *squash_output*)

Build the neural network architecture with the functional API.

Parameters

- `input_shape` (*int*) – Shape of the input features.
- `output_shape` (*int*) – Shape of the output labels.
- `hyperparameters` (*dict*) – Dictionary with the hyperparameter names and values for the neural network outcome prediction model.
- `squash_output` (*bool*) – Indicator for the use of a sigmoid activation function in the output layer.

Returns
Instance of the class *Functional*, which holds methods to make predictions from the neural network model.

Return type
object of class 'Functional'

`compile_and_fit`(*prediction_model*, *features*, *labels*, *hyperparameters*, *validation_data=None*)

Compile and fit the neural network outcome prediction model to the input data.

Parameters

- `prediction_model` (object of class *Functional*) – Instance for the provision of the neural network architecture.
- `features` (*tf.float64*) – Casted array of input feature values.
- `labels` (*tf.float64*) – Casted array of input label values.
- `hyperparameters` (*dict*) – Dictionary with the hyperparameter names and values for the neural network outcome prediction model.
- `validation_data` (*tuple*) – Optional validation features and labels for the fitting procedure.

Returns
prediction_model – Instance of the class *Functional*, which holds methods to make predictions from the neural network model.

Return type
object of class *Functional*

`tune_hyperparameters_with_bayes`(*features*, *labels*)

Tune the hyperparameters of the neural network model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.

Returns
tuned_hyperparameters – Dictionary with the hyperparameter names and values tuned via Bayesian hyperparameter optimization.

Return type
dict

`train`(*features*, *labels*)

Train the neural network outcome prediction model.

Returns
prediction_model – Instance of the class *Functional*, which holds methods to make predictions from the neural network model.

Return type
object of class *Functional*

`predict`(*features*, *squash_output=True*)

Predict the label values from the feature values.

Parameters

- **features** (*ndarray*) – Array of input feature values.
- **squash_output** (*bool*) – Indicator for the use of a sigmoid activation function in the output layer.

Returns
Floating-point label prediction or array of label predictions.

Return type
float or ndarray

`predict_oof`(*features*, *labels*, *oof_splits*)

Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.

Parameters
oof_splits (*int*) – Number of splits for the stratified cross-validation.

Returns
Array with the out-of-folds label predictions.

Return type
ndarray

`inspect`(*labels*)

.

`evaluate`(*features*, *labels*, *oof_splits*)

.

set_file_paths(*base_path*)

Set the paths for model, configuration and hyperparameter files.

Parameters

base_path (*string*) – Base path from which to access the model files.

read_model_from_file()

Read the neural network outcome prediction model from the model file path.

Returns

Instance of the class *Functional*, which holds methods to make predictions from the neural network model.

Return type

object of class *Functional*

write_model_to_file(*prediction_model*)

Write the neural network outcome prediction model to the model file path.

Parameters

prediction_model (object of class *Functional*) – Instance of the class *Functional*, which holds methods to make predictions from the neural network model.

read_configuration_from_file()

Read the configuration dictionary from the configuration file path.

Returns

Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

Return type

dict

write_configuration_to_file(*configuration*)

Write the configuration dictionary to the configuration file path.

Parameters

configuration (*dict*) – Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

read_hyperparameters_from_file()

Read the neural network outcome prediction model hyperparameters from the hyperparameter file path.

Returns

Dictionary with the hyperparameter names and values for the neural network outcome prediction model.

Return type

dict

write_hyperparameters_to_file(*hyperparameters*)

Write the hyperparameter dictionary to the hyperparameter file path.

Parameters

hyperparameters (*dict*) – Dictionary with the hyperparameter names and values for the neural network outcome prediction model.

class pyanno4rt.learning_model.frequentist.**RandomForestModel**(*model_label*, *model_folder_path*, *dataset*, *preprocessing_steps*, *tune_space*, *tune_evaluations*, *tune_score*, *tune_splits*, *inspect_model*, *evaluate_model*, *oof_splits*, *display_options*)

Random forest outcome prediction model class.

This class enables building an individual preprocessing pipeline, fit the random forest model from the input data, inspect the model, make predictions with the model, and assess the predictive performance using multiple evaluation metrics.

The training process includes sequential model-based hyperparameter optimization with tree-structured Parzen estimators and stratified k-fold cross-validation for the objective function evaluation. Cross-validation is also applied to (optionally) inspect the validation feature importances and to generate out-of-folds predictions as a full reconstruction of the input labels for generalization assessment.

Parameters

- **model_label** (*string*) – Label for the random forest model to be used for file naming.
- **dataset** (*dict*) – Dictionary with the raw data set, the label viewpoint, the label bounds, the feature values and names, and the label values and names after modulation. In a compact way, this represents the input data for the random forest model.
- **preprocessing_steps** (*tuple*) – Sequence of labels associated with preprocessing algorithms which make up the preprocessing pipeline for the random forest model. Current available algorithm labels are:
  - transformers : 'Equalizer', 'StandardScaler', 'Whitening'.
- **tune_space** (*dict*) – Search space for the Bayesian hyperparameter optimization, including
  - 'n_estimators' : number of trees in the forest;
  - 'criterion' : measure for the quality of a split;
  - 'max_depth' : maximum depth of each tree;
  - 'min_samples_split' : minimum number of samples required for splitting each node;
  - 'min_samples_leaf' : minimum number of samples required at each node;
  - 'min_weight_fraction_leaf' : minimum weighted fraction of the weights sum required at each node;
  - 'max_features' : maximum number of features taken into account when looking for the best split at each node;
  - 'bootstrap' : indicator for the use of bootstrap samples to build the trees;
  - 'warm_start' : indicator for reusing previous fitting results;
  - 'class_weight' : weights associated with the classes;
  - 'ccp_alpha' : complexity parameter for minimal cost-complexity pruning.
- **tune_evaluations** (*int*) – Number of evaluation steps (trials) for the Bayesian hyperparameter optimization.
- **tune_score** (*string*) – Scoring function for the evaluation of the hyperparameter set candidates. Current available scorers are:

- 'log_loss' : negative log-likelihood score;
- 'roc_auc_score' : area under the ROC curve score.

- **tune_splits** (*int*) – Number of splits for the stratified cross-validation within each hyperparameter optimization step.
- **inspect_model** (*bool*) – Indicator for the inspection of the model, e.g. the feature importances.
- **evaluate_model** (*bool*) – Indicator for the evaluation of the model, e.g. the model KPIs.
- **oof_splits** (*int*) – Number of splits for the stratified cross-validation within the out-of-folds evaluation step of the random forest model.

preprocessor

Instance of the class *DataPreprocessor*, which holds methods to build the preprocessing pipeline, fit with the input features, transform the features, and derive the gradient of the preprocessing algorithms w.r.t the features.

Type
object of class *DataPreprocessor*

features

Values of the input features.

Type
ndarray

labels

Values of the input labels.

Type
ndarray

configuration

Dictionary with information for the modeling, i.e., the dataset, the preprocessing, and the hyperparameter search space.

Type
dict

model_path

Path for storing and retrieving the random forest model.

Type
string

configuration_path

Path for storing and retrieving the configuration dictionary.

Type
string

hyperparameter_path

Path for storing and retrieving the hyperparameter dictionary.

Type
string

`updated_model`

Indicator for the update status of the model, triggers recalculating the model inspection and model evaluation classes.

Type
bool

`prediction_model`

Instance of the class *RandomForestClassifier*, which holds methods to make predictions from the random forest model.

Type
object of class *RandomForestClassifier*

`inspector`

Instance of the class *ModelInspector*, which holds methods to compute model inspection values, e.g. feature importances.

Type
object of class *ModelInspector*

`training_prediction`

Array with the label predictions on the input data.

Type
ndarray

`oof_prediction`

Array with the out-of-folds predictions on the input data.

Type
ndarray

`evaluator`

Instance of the class *ModelEvaluator*, which holds methods to compute the evaluation metrics for a given array with label predictions.

Type
object of class *ModelEvaluator*

Notes

Currently, the preprocessing pipeline for the model is restricted to transformations of the input feature values, e.g. scaling, dimensionality reduction or feature engineering. Transformations which affect the input labels in the same way, e.g. resampling or outlier removal, are not yet possible.

Overview

Table 79: Methods

preprocess(features)	Preprocess the input feature vector with the built pipeline.
get_model(features, labels)	Get the random forest outcome prediction model by reading from the model file path, the datahub, or by training.
tune_hyperparamet labels)	Tune the hyperparameters of the random forest model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.
train(features, labels)	Train the random forest outcome prediction model.
predict(features)	Predict the label values from the feature values.
predict_oof(feature labels, oof_splits)	Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.
inspect(labels)	.
evaluate(features, labels, oof_splits)	.
set_file_paths(bas	Set the paths for model, configuration and hyperparameter files.
read_model_from_f	Read the random forest outcome prediction model from the model file path.
write_model_to_fi	Write the random forest outcome prediction model to the model file path.
read_configuratio	Read the configuration dictionary from the configuration file path.
write_configurati	Write the configuration dictionary to the configuration file path.
read_hyperparamet	Read the random forest outcome prediction model hyperparameters from the hyperparameter file path.
write_hyperparame	Write the hyperparameter dictionary to the hyperparameter file path.

Members

preprocess(*features*)

Preprocess the input feature vector with the built pipeline.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Array of transformed feature values.

Return type
ndarray

get_model(*features*, *labels*)

Get the random forest outcome prediction model by reading from the model file path, the datahub, or by training.

Returns
Instance of the class *RandomForestClassifier*, which holds methods to make predictions from the random forest model.

Return type
object of class *RandomForestClassifier*

tune_hyperparameters_with_bayes(*features*, *labels*)

Tune the hyperparameters of the random forest model via sequential model-based optimization using the

tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.

Returns
tuned_hyperparameters – Dictionary with the hyperparameter names and values tuned via Bayesian hyperparameter optimization.

Return type
dict

`train`(*features*, *labels*)

Train the random forest outcome prediction model.

Returns
prediction_model – Instance of the class *RandomForestClassifier*, which holds methods to make predictions from the random forest model.

Return type
object of class *RandomForestClassifier*

`predict`(*features*)

Predict the label values from the feature values.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Floating-point label prediction or array of label predictions.

Return type
float or ndarray

`predict_oof`(*features*, *labels*, *oof_splits*)

Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.

Parameters
oof_splits (*int*) – Number of splits for the stratified cross-validation.

Returns
Array with the out-of-folds label predictions.

Return type
ndarray

`inspect`(*labels*)

.

`evaluate`(*features*, *labels*, *oof_splits*)

.

`set_file_paths`(*base_path*)

Set the paths for model, configuration and hyperparameter files.

Parameters
base_path (*string*) – Base path from which to access the model files.

`read_model_from_file`()

Read the random forest outcome prediction model from the model file path.

Returns
Instance of the class *RandomForestClassifier*, which holds methods to make predictions from the random forest model.

Return type
object of class *RandomForestClassifier*

`write_model_to_file`(*prediction_model*)

Write the random forest outcome prediction model to the model file path.

Parameters
prediction_model (object of class *RandomForestClassifier*) – Instance of the class *RandomForestClassifier*, which holds methods to make predictions from the random forest model.

`read_configuration_from_file`()

Read the configuration dictionary from the configuration file path.

Returns
Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

Return type
dict

`write_configuration_to_file`(*configuration*)

Write the configuration dictionary to the configuration file path.

Parameters
configuration (*dict*) – Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

`read_hyperparameters_from_file`()

Read the random forest outcome prediction model hyperparameters from the hyperparameter file path.

Returns
Dictionary with the hyperparameter names and values for the random forest outcome prediction model.

Return type
dict

`write_hyperparameters_to_file`(*hyperparameters*)

Write the hyperparameter dictionary to the hyperparameter file path.

Parameters
hyperparameters (*dict*) – Dictionary with the hyperparameter names and values for the random forest outcome prediction model.

`class pyanno4rt.learning_model.frequentist.`**`SupportVectorMachineModel`**(*model_label*, *model_folder_path*, *dataset*, *preprocessing_steps*, *tune_space*, *tune_evaluations*, *tune_score*, *tune_splits*, *inspect_model*, *evaluate_model*, *oof_splits*, *display_options*)

Support vector machine outcome prediction model class.

This class enables building an individual preprocessing pipeline, fit the support vector machine model from the input data, inspect the model, make predictions with the model, and assess the predictive performance using multiple evaluation metrics.

The training process includes sequential model-based hyperparameter optimization with tree-structured Parzen estimators and stratified k-fold cross-validation for the objective function evaluation. Cross-validation is also applied to (optionally) inspect the validation feature importances and to generate out-of-folds predictions as a full reconstruction of the input labels for generalization assessment.

Parameters

- **model_label** (*string*) – Label for the support vector machine model to be used for file naming.
- **dataset** (*dict*) – Dictionary with the raw data set, the label viewpoint, the label bounds, the feature values and names, and the label values and names after modulation. In a compact way, this represents the input data for the support vector machine model.
- **preprocessing_steps** (*tuple*) – Sequence of labels associated with preprocessing algorithms which make up the preprocessing pipeline for the support vector machine model. Current available algorithm labels are:
  - transformers : 'Equalizer', 'StandardScaler', 'Whitening'.
- **tune_space** (*dict*) – Search space for the Bayesian hyperparameter optimization, including
  - 'C' : inverse of the regularization strength;
  - 'kernel' : kernel type for the support vector machine;
  - 'degree' : degree of the polynomial kernel function;
  - 'gamma' : kernel coefficient for RBF, polynomial and sigmoid kernel;
  - 'tol' : tolerance for stopping criteria;
  - 'class_weight' : weights associated with the classes.
- **tune_evaluations** (*int*) – Number of evaluation steps (trials) for the Bayesian hyperparameter optimization.
- **tune_score** (*string*) – Scoring function for the evaluation of the hyperparameter set candidates. Current available scorers are:
  - 'log_loss' : negative log-likelihood score;
  - 'roc_auc_score' : area under the ROC curve score.
- **tune_splits** (*int*) – Number of splits for the stratified cross-validation within each hyperparameter optimization step.
- **inspect_model** (*bool*) – Indicator for the inspection of the model, e.g. the feature importances.
- **evaluate_model** (*bool*) – Indicator for the evaluation of the model, e.g. the model KPIs.
- **oof_splits** (*int*) – Number of splits for the stratified cross-validation within the out-of-folds evaluation step of the support vector machine model.

preprocessor

Instance of the class *DataPreprocessor*, which holds methods to build the preprocessing pipeline, fit with the input features, transform the features, and derive the gradient of the preprocessing algorithms w.r.t the features.

Type
object of class *DataPreprocessor*

`features`

Values of the input features.

Type
ndarray

`labels`

Values of the input labels.

Type
ndarray

`configuration`

Dictionary with information for the modeling, i.e., the dataset, the preprocessing, and the hyperparameter search space.

Type
dict

`model_path`

Path for storing and retrieving the support vector machine model.

Type
string

`configuration_path`

Path for storing and retrieving the configuration dictionary.

Type
string

`hyperparameter_path`

Path for storing and retrieving the hyperparameter dictionary.

Type
string

`updated_model`

Indicator for the update status of the model, triggers recalculating the model inspection and model evaluation classes.

Type
bool

`prediction_model`

Instance of the class *SVC*, which holds methods to make predictions from the support vector machine model.

Type
object of class *SVC*

`inspector`

Instance of the class *ModelInspector*, which holds methods to compute model inspection values, e.g. feature importances.

Type
object of class *ModelInspector*

training_prediction

Array with the label predictions on the input data.

Type
ndarray

oof_prediction

Array with the out-of-folds predictions on the input data.

Type
ndarray

evaluator

Instance of the class *ModelEvaluator*, which holds methods to compute the evaluation metrics for a given array with label predictions.

Type
object of class *ModelEvaluator*

Notes

Currently, the preprocessing pipeline for the model is restricted to transformations of the input feature values, e.g. scaling, dimensionality reduction or feature engineering. Transformations which affect the input labels in the same way, e.g. resampling or outlier removal, are not yet possible.

Overview

Table 80: Methods

preprocess(features	Preprocess the input feature vector with the built pipeline.
get_model(features, labels)	Get the support vector machine outcome prediction model by reading from the model file path, the datahub, or by training.
tune_hyperparame labels)	Tune the hyperparameters of the support vector machine model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.
train(features, labels)	Train the support vector machine outcome prediction model.
predict(features)	Predict the label values from the feature values.
predict_oof(featur labels, oof_splits)	Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.
inspect(labels)	.
evaluate(features, labels, oof_splits)	.
set_file_paths(ba	Set the paths for model, configuration and hyperparameter files.
*read_model_from_*	Read the support vector machine outcome prediction model from the model file path.
write_model_to_f	Write the support vector machine outcome prediction model to the model file path.
read_configurati	Read the configuration dictionary from the configuration file path.
write_configurat	Write the configuration dictionary to the configuration file path.
read_hyperparame	Read the support vector machine outcome prediction model hyperparameters from the hyperparameter file path.
write_hyperparam	Write the hyperparameter dictionary to the hyperparameter file path.

Members

preprocess(*features*)

Preprocess the input feature vector with the built pipeline.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Array of transformed feature values.

Return type
ndarray

get_model(*features*, *labels*)

Get the support vector machine outcome prediction model by reading from the model file path, the datahub, or by training.

Returns
Instance of the class *SVC*, which holds methods to make predictions from the support vector machine model.

Return type
object of class *SVC*

tune_hyperparameters_with_bayes(*features*, *labels*)

Tune the hyperparameters of the support vector machine model via sequential model-based optimization using the tree-structured Parzen estimator. As a variation, the objective function is evaluated based on a stratified k-fold cross-validation.

Returns
tuned_hyperparameters – Dictionary with the hyperparameter names and values tuned via Bayesian hyperparameter optimization.

Return type
dict

train(*features*, *labels*)

Train the support vector machine outcome prediction model.

Returns
prediction_model – Instance of the class *SVC*, which holds methods to make predictions from the support vector machine model.

Return type
object of class *SVC*

predict(*features*)

Predict the label values from the feature values.

Parameters
features (*ndarray*) – Array of input feature values.

Returns
Floating-point label prediction or array of label predictions.

Return type
float or ndarray

predict_oof(*features*, *labels*, *oof_splits*)

Predict the out-of-folds (OOF) labels using a stratified k-fold cross-validation.

Parameters
oof_splits (*int*) – Number of splits for the stratified cross-validation.

Returns
Array with the out-of-folds label predictions.

Return type
ndarray

inspect(*labels*)

.

evaluate(*features*, *labels*, *oof_splits*)

.

set_file_paths(*base_path*)

Set the paths for model, configuration and hyperparameter files.

Parameters
base_path (*string*) – Base path from which to access the model files.

read_model_from_file()

Read the support vector machine outcome prediction model from the model file path.

Returns
Instance of the class *SVC*, which holds methods to make predictions from the support vector machine model.

Return type
object of class *SVC*

write_model_to_file(*prediction_model*)

Write the support vector machine outcome prediction model to the model file path.

Parameters
prediction_model (object of class *SVC*) – Instance of the class *SVC*, which holds methods to make predictions from the support vector machine model.

read_configuration_from_file()

Read the configuration dictionary from the configuration file path.

Returns
Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

Return type
dict

write_configuration_to_file(*configuration*)

Write the configuration dictionary to the configuration file path.

Parameters
configuration (*dict*) – Dictionary with information for the modeling, i.e., the dataset, the preprocessing steps, and the hyperparameter search space.

read_hyperparameters_from_file()

Read the support vector machine outcome prediction model hyperparameters from the hyperparameter file path.

Returns
Dictionary with the hyperparameter names and values for the support vector machine outcome prediction model.

Return type
dict

`write_hyperparameters_to_file`(*hyperparameters*)

Write the hyperparameter dictionary to the hyperparameter file path.

Parameters
hyperparameters (*dict*) – Dictionary with the hyperparameter names and values for the support vector machine outcome prediction model.

pyanno4rt.learning_model.inspection

Model inspection module.

The module aims to provide methods and classes to inspect the applied learning models.

Subpackages

pyanno4rt.learning_model.inspection.inspections

Inspection algorithms module.

The module aims to provide methods and classes to inspect the applied learning models.

Overview

Table 81: Classes

PermutationImportance	Permutation importance class.

Classes

class pyanno4rt.learning_model.inspection.inspections.**PermutationImportance**(*model_name*, *model_class*, *hyperparameters=None*)

Permutation importance class.

Parameters

- **model_name** (*string*) – Name of the learning model.
- **hyperparameters** (*dict, default = None*) – Hyperparameters dictionary.

model_name

See 'Parameters'.

Type
string

hyperparameters

See 'Parameters'.

Type
dict

Overview

Table 82: Methods

compute(model, features, labels, number_of_repeats)	Compute the training permutation importance.
compute_oof(model, features, labels, number_of_repeats)	Compute the validation permutation importance.
score(model, features, true_labels)	Create a callable for scoring the model.

Members

compute(*model*, *features*, *labels*, *number_of_repeats*)

Compute the training permutation importance.

Parameters

- **model** (*object*) – Instance of the outcome prediction model.
- **features** (*ndarray*) – Values of the input features.
- **labels** (*ndarray*) – Values of the input labels.
- **number_of_repeats** (*int*) – Number of feature permutations to evaluate.

Returns
Permutation importance values per repetition.

Return type
ndarray

compute_oof(*model*, *features*, *labels*, *number_of_repeats*)

Compute the validation permutation importance.

Parameters

- **model** (*object*) – Instance of the outcome prediction model.
- **features** (*ndarray*) – Values of the input features.
- **labels** (*ndarray*) – Values of the input labels.
- **number_of_repeats** (*int*) – Number of feature permutations to evaluate.

Returns
Permutation importance values per repetition and fold.

Return type
tuple

score(*model*, *features*, *true_labels*)

Create a callable for scoring the model.

Parameters

- **model** (*object*) – Instance of the outcome prediction model.
- **features** (*ndarray*) – Values of the input features.
- **labels** (*ndarray*) – Values of the input labels.

Returns
Score of the loss function.

Return type
float

Overview

Table 83: Classes

ModelInspector	Model inspection class.

Classes

class pyanno4rt.learning_model.inspection.**ModelInspector**(*model_name*, *model_class*, *hyperparameters=None*)

Model inspection class.

This class provides a collection of inspection methods to be computed in a single method call.

Parameters

- **model_name** (*string*) – Name of the learning model.
- **hyperparameters** (*dict*) – Hyperparameters dictionary.

model_name

See 'Parameters'.

Type
string

hyperparameters

See 'Parameters'.

Type
dict, default = None

inspections

Dictionary with the inspection values.

Type
dict

Overview

Table 84: Methods

compute(model, features, labels, number_of_repeats)	Compute the inspection results.

Members

compute(*model*, *features*, *labels*, *number_of_repeats*)

Compute the inspection results.

Parameters

- **model** (*object*) – Instance of the outcome prediction model.
- **features** (*ndarray*) – Values of the input features.
- **labels** (*ndarray*) – Values of the input labels.
- **number_of_repeats** (*int*) – Number of feature permutations to evaluate.

pyanno4rt.learning_model.losses

Losses module.

The module aims to provide loss functions to support the model training.

Overview

Table 85: Function

brier_loss(true_labels, predicted_labels)	Compute the log loss from the true and predicted labels.
log_loss(true_labels, predicted_labels)	Compute the log loss from the true and predicted labels.

Functions

pyanno4rt.learning_model.losses.**brier_loss**(*true_labels*, *predicted_labels*)

Compute the log loss from the true and predicted labels.

pyanno4rt.learning_model.losses.**log_loss**(*true_labels*, *predicted_labels*)

Compute the log loss from the true and predicted labels.

pyanno4rt.learning_model.preprocessing

Data preprocessing module.

The module aims to provide methods and classes for data preprocessing, i.e., for building up a flexible preprocessing pipeline with data cleaning, reduction, and transformation algorithms.

Subpackages

pyanno4rt.learning_model.preprocessing.cleaners

Cleaners module.

The module aims to provide methods and classes for data cleaning in the context of data preprocessing.

pyanno4rt.learning_model.preprocessing.reducers

Reducers module.

The module aims to provide methods and classes for data reduction in the context of data preprocessing.

pyanno4rt.learning_model.preprocessing.samplers

Samplers module.

The module aims to provide methods and classes for data sampling in the context of data preprocessing.

pyanno4rt.learning_model.preprocessing.transformers

Transformers module.

The module aims to provide methods and classes for data transformation in the context of data preprocessing.

Overview

Table 86: Classes

Equalizer	Equalizer transformer class.
StandardScaler	Standard scaling transformer class.
Whitening	Whitening transformer class.

Classes

class pyanno4rt.learning_model.preprocessing.transformers.**Equalizer**

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Equalizer transformer class.

This class provides methods to propagate the input features in an unchanged matter, i.e., to build a "neutral" preprocessing pipeline.

Overview

Table 87: Attributes

label	-

Table 88: Methods

fit(features, labels)	Fit the transformator with the input data.
transform(features, labels)	Transform the input data.
compute_gradient(feature	Compute the gradient of the equalization transformation with respect to the features.

Members

label = 'Equalizer'

fit(*features*, *labels=None*)

Fit the transformator with the input data.

Parameters

- **features** (*ndarray*)
- **features.** (*Values of the input*)
- **labels** (*ndarray, default = None*) – Values of the input labels.

transform(*features*, *labels=None*)

Transform the input data.

Parameters

- **features** (*ndarray*)
- **features.** (*Values of the input*)
- **labels** (*ndarray, default = None*) – Values of the input labels.

compute_gradient(*features*)

Compute the gradient of the equalization transformation with respect to the features.

Parameters

features (*ndarray*) – Values of the input features.

Returns
Values of the input feature gradients.

Return type
ndarray

class pyanno4rt.learning_model.preprocessing.transformers.**StandardScaler**(*center=True*, *scale=True*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Standard scaling transformer class.

This class provides methods to fit the standard scaler, transform input features, and return the scaler gradient.

Parameters

- **center** (*bool*) – Indicator for the computation of the mean values and centering of the data.
- **scale** (*bool*) – Indicator for the computation of the standard deviations and the scaling of the data.

center

See 'Parameters'.

Type
bool

scale

See 'Parameters'.

Type
bool

means

Mean values of the feature columns. Only computed if ``center``is set to True, otherwise it is set to zeros.

Type
ndarray

deviations

Standard deviations of the feature columns. Only computed if ``scale``is set to True, otherwise it is set to ones

Type
ndarray

Overview

Table 89: Attributes

label	-

Table 90: Methods

fit(features, labels)	Fit the transformator with the input data.
transform(features, labels)	Transform the input data.
compute_gradient(featur	Compute the gradient of the standard scaling transformation with respect to the features.

Members

label = 'StandardScaler'

fit(*features*, *labels=None*)

Fit the transformator with the input data.

Parameters

- **features** (*ndarray*)
- **features.** (*Values of the input*)
- **labels** (*ndarray, default = None*) – Values of the input labels.

transform(*features*, *labels=None*)

Transform the input data.

Parameters

- **features** (*ndarray*)
- **features.** (*Values of the input*)
- **labels** (*ndarray, default = None*) – Values of the input labels.

Returns
Values of the transformed features.

Return type
ndarray

compute_gradient(*features*)

Compute the gradient of the standard scaling transformation with respect to the `features`.

Parameters
features (*ndarray*) – Values of the input features.

Returns
Values of the input feature gradients.

Return type
ndarray

class pyanno4rt.learning_model.preprocessing.transformers.**Whitening**(*method='zca'*)

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.TransformerMixin`

Whitening transformer class.

This class provides methods to fit the whitening matrix, transform input features, and return the whitening gradient.

Parameters
method (*{'pca', 'zca'}, default = 'zca'*) – Method for the computation of the whitening matrix. With 'zca', the zero-phase component analysis (or Mahalanobis transformation) is applied, with 'pca', the principal component analysis lays the groundwork.

method

See 'Parameters'.

Type
{'pca', 'zca'}

means

Mean values of the feature columns.

Type
ndarray

matrix

Whitening matrix for the transformation of feature vectors.

Type
ndarray

Overview

Table 91: Attributes

label	-

Table 92: Methods

fit(features, labels)	Fit the transformator with the input data.
transform(features, labels)	Transform the input data.
compute_gradient(features	Compute the gradient of the whitening transformation with respect to the features.

Members

label = 'Whitening'

fit(*features*, *labels=None*)

Fit the transformator with the input data.

Parameters

- **features** (*ndarray*)
- **features.** (*Values of the input*)
- **labels** (*ndarray, default = None*) – Values of the input labels.

transform(*features*, *labels=None*)

Transform the input data.

Parameters

- **features** (*ndarray*)
- **features.** (*Values of the input*)
- **labels** (*ndarray, default = None*) – Values of the input labels.

Returns
Values of the transformed features.

Return type
ndarray

compute_gradient(*features*)

Compute the gradient of the whitening transformation with respect to the `features`.

Parameters

features (*ndarray*) – Values of the input features.

Returns

Values of the input feature gradients.

Return type

ndarray

Overview

Table 93: Classes

DataPreprocessor	Data preprocessing pipeline class.

Classes

class pyanno4rt.learning_model.preprocessing.**DataPreprocessor**(*step_labels*, *verbose=True*)

Data preprocessing pipeline class.

Parameters

step_labels (*tuple*) – Tuple with the preprocessing pipeline elements (labels of the respective preprocessing algorithm classes).

labels

Tuple with the step labels.

Type

tuple

steps

Tuple with the preprocessing algorithms.

Type

tuple

pipeline

Instance of the class *Pipeline*, which provides a preprocessing pipeline (chain of transformation algorithms).

Type

object of class *Pipeline*

Overview

Table 94: Methods

build(verbose)	Build the preprocessing pipeline from the passed steps and step labels.
fit(features)	Fit the preprocessing pipeline with the input features.
transform(features)	Transform the input features with the preprocessing pipeline.
fit_transform(features)	Fit and transform the input features with the preprocessing pipeline.
gradientize(features)	Compute the gradient of the preprocessing pipeline w.r.t the input features.

Members

build(*verbose*)

Build the preprocessing pipeline from the passed steps and step labels.

Returns
Instance of the class *Pipeline*, which provides a preprocessing pipeline (chain of transformation algorithms).

Return type
object of class *Pipeline*

fit(*features*)

Fit the preprocessing pipeline with the input features.

Parameters
features (*ndarray*) – Values of the input features.

transform(*features*)

Transform the input features with the preprocessing pipeline.

Returns
Array of transformed feature values.

Return type
ndarray

fit_transform(*features*)

Fit and transform the input features with the preprocessing pipeline.

Parameters
features (*ndarray*) – Values of the input features.

Returns
Array of transformed feature values.

Return type
ndarray

gradientize(*features*)

Compute the gradient of the preprocessing pipeline w.r.t the input features.

Parameters
features (*ndarray*) – Array of transformed feature values.

Returns
Input feature gradients for the full preprocessing pipeline.

Return type
list

Overview

Table 95: Classes

DataModelHandler	Data & learning model handling class.

Classes

class pyanno4rt.learning_model.**DataModelHandler**(*model_label*, *data_path*, *feature_filter*, *label_name*, *label_bounds*, *time_variable_name*, *label_viewpoint*, *fuzzy_matching*, *write_features*)

Data & learning model handling class.

This class implements methods to handle the integration of the base dataset, the feature map generator and the feature calculator.

Parameters

- **data_path** (*str*) – Path to the data set used for fitting the machine learning model.
- **feature_filter** (*dict, default={'features': [], 'filter_mode': 'remove'}*) – Dictionary with a list of feature names and a value from {'retain', 'remove'} as an indicator for retaining/removing the features prior to model fitting.
- **label_bounds** (*list, default=[1, 1]*) – Bounds for the label values to binarize into positive (value lies inside the bounds) and negative class (value lies outside the bounds).
- **label_viewpoint** (*{'early', 'late', 'long-term', 'longitudinal', 'profile'}, default='long-term'*) – Time of observation for the presence of tumor control and/or normal tissue complication events.
- **fuzzy_matching** (*bool, default=True*) – Indicator for the use of fuzzy string matching to generate the feature map (if False, exact string matching is applied).
- **write_features** (*bool, default=True*) – Indicator for writing the iteratively calculated feature vectors into a feature history.

model_label

See 'Parameters'.

Type
str

data_path

See 'Parameters'.

Type
str

write_features

See 'Parameters'.

Type
bool

dataset

The object used to handle the base dataset.

Type

object of class `TabularDataGenerator`

feature_map_generator

The object used to map the dataset features to the feature definitions.

Type

object of class `FeatureMapGenerator`

feature_calculator

The object used to (re-)calculate the feature values and gradients.

Type

object of class `FeatureCalculator`

Overview

Table 96: Methods

integrate()	Integrate the learning model-related classes.
process_feature_history()	Process the feature history from the feature calculator.

Members

integrate()

Integrate the learning model-related classes.

process_feature_history()

Process the feature history from the feature calculator.

pyanno4rt.logging

Logging module.

This module aims to provide methods and classes to configure an instance of the logger.

Overview

Table 97: Classes

Logger	Logging class.

Classes

class pyanno4rt.logging.**Logger**(**args*)

Logging class.

This class provides methods to configure an instance of the logger, including multiple stream handlers and formatters to print messages at different levels.

Parameters

***args** (*tuple*) – Tuple with optional (non-keyworded) logging parameters. The value args[0] refers to the label of the treatment plan, while args[1] specifies the minimum logging level.

logger

The external object used to interface the logging methods.

Type

object of class `Logger`

Overview

Table 98: Methods

initialize_logger(label, min_log_level)	Initialize the logger by specifying the channel name, handlers, and formatters.
display_to_console(level, formatted_string, *args)	Call the display function specified by *level* for the message given by *formatted_string*.
display_debug(formatted_string, *args)	Display a logging message for the level 'debug'.
display_info(formatted_string, *args)	Display a logging message for the level 'info'.
display_warning(formatted_string, *args)	Display a logging message for the level 'warning'.
display_error(formatted_string, *args)	Display a logging message for the level 'error'.
display_critical(formatted_string, *args)	Display a logging message for the level 'critical'.

Members

initialize_logger(*label*, *min_log_level*)

Initialize the logger by specifying the channel name, handlers, and formatters.

Parameters

- **label** (*str*) – Label of the treatment plan instance.
- **min_log_level** (*{'debug', 'info', 'warning', 'error', 'critical'}*) – Minimum logging level for broadcasting messages to the console and the object streams.

display_to_console(*level*, *formatted_string*, **args*)

Call the display function specified by *level* for the message given by *formatted_string*.

Parameters

- **level** (*{'debug', 'info', 'warning', 'error', 'critical'}*) – Level of the logging message.

- **formatted_string** (*str*) – Formatted string to be displayed.
- ***args** (*tuple*) – Optional display parameters.

display_debug(*formatted_string*, **args*)

Display a logging message for the level 'debug'.

Parameters

- **formatted_string** (*str*) – Formatted string to be displayed.
- ***args** (*tuple*) – Optional display parameters.

display_info(*formatted_string*, **args*)

Display a logging message for the level 'info'.

Parameters

- **formatted_string** (*str*) – Formatted string to be displayed.
- ***args** (*tuple*) – Optional display parameters.

display_warning(*formatted_string*, **args*)

Display a logging message for the level 'warning'.

Parameters

- **formatted_string** (*str*) – Formatted string to be displayed.
- ***args** (*tuple*) – Optional display parameters.

display_error(*formatted_string*, **args*)

Display a logging message for the level 'error'.

Parameters

- **formatted_string** (*str*) – Formatted string to be displayed.
- ***args** (*tuple*) – Optional display parameters.

display_critical(*formatted_string*, **args*)

Display a logging message for the level 'critical'.

Parameters

- **formatted_string** (*str*) – Formatted string to be displayed.
- ***args** (*tuple*) – Optional display parameters.

pyanno4rt.optimization

Optimization module.

This module aims to provide methods and classes for setting up and solving the inverse planning problem.

Subpackages

pyanno4rt.optimization.components

Components module.

The module aims to provide methods and classes to handle dose-related and outcome model-based component functions for the optimization problem.

Overview

Table 99: Classes

ConventionalComponentClass	Conventional component template class.
MachineLearningComponentClass	Machine learning component template class.
RadiobiologyComponentClass	Radiobiology component template class.
DecisionTreeNTCP	Decision tree NTCP component class.
DecisionTreeTCP	Decision tree TCP component class.
DoseUniformity	Dose uniformity component class.
EquivalentUniformDose	Equivalent uniform dose (EUD) component class.
KNeighborsNTCP	K-nearest neighbors NTCP component class.
KNeighborsTCP	K-nearest neighbors TCP component class.
LogisticRegressionNTCP	Logistic regression NTCP component class.
LogisticRegressionTCP	Logistic regression TCP component class.
LQPoissonTCP	Linear-quadratic Poisson TCP component class.
LymanKutcherBurmanNTCP	Lyman-Kutcher-Burman (LKB) NTCP component class.
MaximumDVH	Maximum dose-volume histogram (Maximum DVH) component class.
MeanDose	Mean dose component class.
MinimumDVH	Minimum dose-volume histogram (Minimum DVH) component class.
NaiveBayesNTCP	Naive Bayes NTCP component class.
NaiveBayesTCP	Naive Bayes TCP component class.
NeuralNetworkNTCP	Neural network NTCP component class.
NeuralNetworkTCP	Neural network TCP component class.
RandomForestNTCP	Random forest NTCP component class.
RandomForestTCP	Random forest TCP component class.
SquaredDeviation	Squared deviation component class.
SquaredOverdosing	Squared overdosing component class.
SquaredUnderdosing	Squared underdosing component class.
SupportVectorMachineNTCP	Support vector machine NTCP component class.
SupportVectorMachineTCP	Support vector machine TCP component class.

Table 100: Attributes

component_map	-

Classes

class pyanno4rt.optimization.components.**ConventionalComponentClass**(*name*, *parameter_name*, *parameter_category*, *parameter_value*, *embedding*, *weight*, *bounds*, *link*, *identifier*, *display*)

Conventional component template class.

Parameters

- **name** (*str*) – Name of the component class.
- **parameter_name** (*tuple*) – Name of the component parameters.
- **parameter_category** (*tuple*) – Category of the component parameters.
- **parameter_value** (*tuple*) – Value of the component parameters.
- **embedding** (*{'active', 'passive'}*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float*) – Weight of the component function.
- **bounds** (*None or list*) – Constraint bounds for the component.
- **link** (*None or list*) – Other segments used for joint evaluation.
- **identifier** (*str*) – Additional string for naming the component.
- **display** (*bool*) – Indicator for the display of the component.

name

See 'Parameters'.

Type

str

parameter_name

See 'Parameters'.

Type

tuple

parameter_category

See 'Parameters'.

Type

tuple

parameter_value

See 'Parameters'.

Type

list

embedding

See 'Parameters'.

Type

{'active', 'passive'}

weight

See 'Parameters'.

Type
float

bounds

See 'Parameters'.

Type
list

link

See 'Parameters'.

Type
list

identifier

See 'Parameters'.

Type
str

display

See 'Parameters'.

Type
bool

adjusted_parameters

Indicator for the adjustment of the parameters due to fractionation.

Type
bool

RETURNS_OUTCOME

Indicator for the outcome focus of the component.

Type
bool

DEPENDS_ON_MODEL

Indicator for the model dependency of the component.

Type
bool

Overview

Table 101: Methods

get_parameter_value()	Get the value of the parameters.
set_parameter_value(*args)	Set the value of the parameters.
get_weight_value()	Get the value of the weight.
set_weight_value(*args)	Set the value of the weight.
compute_value(*args)	abc Compute the component value.
compute_gradient(*args)	abc Compute the component gradient.

Members

get_parameter_value()

Get the value of the parameters.

Returns
Value of the parameters.

Return type
list

set_parameter_value(**args*)

Set the value of the parameters.

Parameters
***args** (*tuple*) – Keyworded parameters. args[0] should give the value to be set.

get_weight_value()

Get the value of the weight.

Returns
Value of the weight.

Return type
float

set_weight_value(**args*)

Set the value of the weight.

Parameters
***args** (*tuple*) – Keyworded parameters. args[0] should give the value to be set.

abstract compute_value(**args*)

Compute the component value.

abstract compute_gradient(**args*)

Compute the component gradient.

class pyanno4rt.optimization.components.**MachineLearningComponentClass**(*name*, *parameter_name*, *parameter_category*, *model_parameters*, *embedding*, *weight*, *bounds*, *link*, *identifier*, *display*)

Machine learning component template class.

Parameters

- **name** (*str*) – Name of the component class.
- **parameter_name** (*tuple*) – Name of the component parameters.
- **parameter_category** (*tuple*) – Category of the component parameters.
- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters:
  - **model_label**
    [str] Label for the machine learning model.
  - **model_folder_path**
    [str] Path to a folder for loading an external machine learning model.

- **data_path**
  [str] Path to the data set used for fitting the machine learning model.
- **feature_filter**
  [dict, default={'features': [], 'filter_mode': 'remove'}] Dictionary with a list of feature names and a value from {'retain', 'remove'} as an indicator for retaining/removing the features prior to model fitting.
- **label_name**
  [str] Name of the label variable.
- **label_bounds**
  [list, default=[1, 1]] Bounds for the label values to binarize into positive (value lies inside the bounds) and negative class (value lies outside the bounds).
- **time_variable_name**
  [str, default=None] Name of the time-after-radiotherapy variable (unit should be days).
- **label_viewpoint**
  [{'early', 'late', 'long-term', 'longitudinal', 'profile'}, default='long-term'] Time of observation for the presence of tumor control and/or normal tissue complication events. The options can be described as follows:
  * 'early' : event between 0 and 6 months after treatment
  * 'late' : event between 6 and 15 months after treatment
  * 'long-term' : event between 15 and 24 months after treatment
  * 'longitudinal' : no period, time after treatment as covariate
  * 'profile' : TCP/NTCP profiling over time, multi-label scenario with one label per month (up to 24 labels in total).
- **fuzzy_matching**
  [bool, default=True] Indicator for the use of fuzzy string matching to generate the feature map (if False, exact string matching is applied).
- **preprocessing_steps**
  [list, default=['Equalizer']] Sequence of labels associated with preprocessing algorithms to preprocess the input features.

  The following preprocessing steps are currently available:
  * 'Equalizer' `Equalizer`
  * 'StandardScaler' `StandardScaler`
  * 'Whitening' `Whitening`
- **architecture**
  [{'input-convex', 'standard'}, default='input-convex'] Type of architecture for the neural network model.
- **max_hidden_layers**
  [int, default=2] Maximum number of hidden layers for the neural network model.
- **tune_space**
  [dict, default={}] Search space for the Bayesian hyperparameter optimization.

- **tune_evaluations**
  [int, default=50] Number of evaluation steps (trials) for the Bayesian hyperparameter optimization.
- **tune_score**
  [{'AUC', 'Brier score', 'Logloss'}, default='Logloss'] Scoring function for the evaluation of the hyperparameter set candidates.
- **tune_splits**
  [int, default=5] Number of splits for the stratified cross-validation within each hyperparameter optimization step.
- **inspect_model**
  [bool, default=False] Indicator for the inspection of the machine learning model.
- **evaluate_model**
  [bool, default=False] Indicator for the evaluation of the machine learning model.
- **oof_splits**
  [int, default=5] Number of splits for the stratified cross-validation within the out-of-folds evaluation step.
- **write_features**
  [bool, default=True] Indicator for writing the iteratively calculated feature vectors into a feature history.
- **display_options**
  [dict, default={'graphs': ['AUC-ROC', 'AUC-PR', 'F1'], 'kpis': ['Logloss', 'Brier score', 'Subset accuracy', 'Cohen Kappa', 'Hamming loss', 'Jaccard score', 'Precision', 'Recall', 'F1 score', 'MCC', 'AUC']}] Dictionary with the graph and KPI display options.

- **embedding** (*{'active', 'passive'}*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float*) – Weight of the component function.
- **bounds** (*None or list*) – Constraint bounds for the component.
- **link** (*None or list*) – Other segments used for joint evaluation.
- **identifier** (*str*) – Additional string for naming the component.
- **display** (*bool*) – Indicator for the display of the component.

name

See 'Parameters'.

Type
str

parameter_name

See 'Parameters'.

Type
tuple

parameter_category

See 'Parameters'.

Type
tuple

`parameter_value`
Value of the component parameters.

Type
list

`embedding`
See 'Parameters'.

Type
{'active', 'passive'}

`weight`
See 'Parameters'.

Type
float

`bounds`
See 'Parameters'.

Type
list

`link`
See 'Parameters'.

Type
list

`identifier`
See 'Parameters'.

Type
str

`display`
See 'Parameters'.

Type
bool

`model_parameters`
See 'Parameters'.

Type
dict

`adjusted_parameters`
Indicator for the adjustment of the parameters due to fractionation.

Type
bool

`RETURNS_OUTCOME`
Indicator for the outcome focus of the component.

Type
bool

DEPENDS_ON_MODEL

Indicator for the model dependency of the component.

Type
bool

Overview

Table 102: Methods

get_parameter_value()	Get the value of the parameters.
set_parameter_value(*args)	Set the value of the parameters.
get_weight_value()	Get the value of the weight.
set_weight_value(*args)	Set the value of the weight.
add_model()	abc Add the machine learning model to the component.
compute_value(*args)	abc Compute the component value.
compute_gradient(*args)	abc Compute the component gradient.

Members

get_parameter_value()

Get the value of the parameters.

Returns
Value of the parameters.

Return type
list

set_parameter_value(**args*)

Set the value of the parameters.

Parameters
***args** (*tuple*) – Keyworded parameters. args[0] should give the value to be set.

get_weight_value()

Get the value of the weight.

Returns
Value of the weight.

Return type
float

set_weight_value(**args*)

Set the value of the weight.

Parameters
***args** (*tuple*) – Keyworded parameters. args[0] should give the value to be set.

abstract add_model()

Add the machine learning model to the component.

abstract compute_value(**args*)

Compute the component value.

abstract compute_gradient(**args*)

Compute the component gradient.

class pyanno4rt.optimization.components.**RadiobiologyComponentClass**(*name*, *parameter_name*, *parameter_category*, *parameter_value*, *embedding*, *weight*, *bounds*, *link*, *identifier*, *display*)

Radiobiology component template class.

Parameters

- **name** (*str*) – Name of the component class.
- **parameter_name** (*tuple*) – Name of the component parameters.
- **parameter_category** (*tuple*) – Category of the component parameters.
- **parameter_value** (*tuple*) – Value of the component parameters.
- **embedding** (*{'active', 'passive'}*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float*) – Weight of the component function.
- **bounds** (*None or list*) – Constraint bounds for the component.
- **link** (*None or list*) – Other segments used for joint evaluation.
- **identifier** (*str*) – Additional string for naming the component.
- **display** (*bool*) – Indicator for the display of the component.

name

See 'Parameters'.

Type

str

parameter_name

See 'Parameters'.

Type

tuple

parameter_category

See 'Parameters'.

Type

tuple

parameter_value

See 'Parameters'.

Type

list

embedding

See 'Parameters'.

Type

{'active', 'passive'}

weight

See 'Parameters'.

Type
float

bounds

See 'Parameters'.

Type
list

link

See 'Parameters'.

Type
list

identifier

See 'Parameters'.

Type
str

display

See 'Parameters'.

Type
bool

adjusted_parameters

Indicator for the adjustment of the parameters due to fractionation.

Type
bool

RETURNS_OUTCOME

Indicator for the outcome focus of the component.

Type
bool

DEPENDS_ON_MODEL

Indicator for the model dependency of the component.

Type
bool

Overview

Table 103: Methods

get_parameter_value()	Get the value of the parameters.
set_parameter_value(*args)	Set the value of the parameters.
get_weight_value()	Get the value of the weight.
set_weight_value(*args)	Set the value of the weight.
compute_value(*args)	abc Compute the component value.
compute_gradient(*args)	abc Compute the component gradient.

Members

get_parameter_value()

Get the value of the parameters.

Returns
Value of the parameters.

Return type
list

set_parameter_value(**args*)

Set the value of the parameters.

Parameters
***args** (*tuple*) – Keyworded parameters. args[0] should give the value to be set.

get_weight_value()

Get the value of the weight.

Returns
Value of the weight.

Return type
float

set_weight_value(**args*)

Set the value of the weight.

Parameters
***args** (*tuple*) – Keyworded parameters. args[0] should give the value to be set.

abstract compute_value(**args*)

Compute the component value.

abstract compute_gradient(**args*)

Compute the component gradient.

class pyanno4rt.optimization.components.**DecisionTreeNTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Decision tree NTCP component class.

This class provides methods to compute the value and the gradient of the decision tree NTCP component, as well as to add the decision tree model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.

- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the decision tree model.

Type
object of class `DecisionTreeModel`

parameter_value

Value of the decision tree model parameters.

Type
list

bounds

See 'Parameters'.

Type
list

Overview

Table 104: Methods

add_model()	Add the decision tree model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the decision tree model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**DecisionTreeTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Decision tree TCP component class.

This class provides methods to compute the value and the gradient of the decision tree TCP component, as well as to add the decision tree model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type

object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the decision tree model.

Type

object of class `DecisionTreeModel`

parameter_value

Value of the decision tree model parameters.

Type

list

bounds

See 'Parameters'.

Type

list

Overview

Table 105: Methods

add_model()	Add the decision tree model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the decision tree model to the component.

compute_value(**args*)

Compute the component value.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component function.

Return type

float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**DoseUniformity**(*embedding='active', weight=1.0, bounds=None, link=None, identifier=None, display=True*)

Bases: *pyanno4rt.optimization.components.ConventionalComponentClass*

Dose uniformity component class.

This class provides methods to compute the value and the gradient of the dose uniformity component.

Parameters

- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.

- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 106: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**EquivalentUniformDose**(*target_eud=None, volume_parameter=None, embedding='active', weight=1.0, bounds=None, link=None, identifier=None, display=True*)

Bases: *pyanno4rt.optimization.components.ConventionalComponentClass*

Equivalent uniform dose (EUD) component class.

This class provides methods to compute the value and the gradient of the EUD component.

Parameters

- **target_eud** (*int or float, default=None*) – Target value for the EUD.

- **volume_parameter** (*int or float, default=None*) – Dose-volume effect parameter.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 107: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**KNeighborsNTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

K-nearest neighbors NTCP component class.

This class provides methods to compute the value and the gradient of the k-nearest neighbors NTCP component, as well as to add the k-nearest neighbors model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the k-nearest neighbors model.

Type
object of class `KNeighborsModel`

parameter_value

Value of the k-nearest neighbors model parameters.

Type
list

bounds

See 'Parameters'.

Type
list

Overview

Table 108: Methods

add_model()	Add the k-nearest neighbors model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the k-nearest neighbors model to the component.

compute_value(**args*)

Compute the component value.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component function.

Return type

float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**KNeighborsTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

K-nearest neighbors TCP component class.

This class provides methods to compute the value and the gradient of the k-nearest neighbors TCP component, as well as to add the k-nearest neighbors model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.

- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the k-nearest neighbors model.

Type
object of class `KNeighborsModel`

parameter_value

Value of the k-nearest neighbors model parameters.

Type
list

bounds

See 'Parameters'.

Type
list

Overview

Table 109: Methods

add_model()	Add the k-nearest neighbors model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the k-nearest neighbors model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**LogisticRegressionNTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Logistic regression NTCP component class.

This class provides methods to compute the value and the gradient of the logistic regression NTCP component, as well as to add the logistic regression model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class MachineLearningComponentClass.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type

object of class DataModelHandler

model

The object used to preprocess, tune, train, inspect and evaluate the logistic regression model.

Type

object of class LogisticRegressionModel

parameter_value

Value of the logistic regression model coefficients.

Type

list

intercept_value

Value of the logistic regression model intercept.

Type
list

bounds

See 'Parameters'. Transformed by the inverse sigmoid function.

Type
list

Overview

Table 110: Methods

get_intercept_value()	Get the value of the intercept.
set_intercept_value(*args)	Set the value of the intercept.
add_model()	Add the logistic regression model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

get_intercept_value()

Get the value of the intercept.

Returns
Value of the intercept.

Return type
list

set_intercept_value(**args*)

Set the value of the intercept.

Parameters
***args** (*tuple*) – Keyworded parameters. args[0] should give the value to be set.

add_model()

Add the logistic regression model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**LogisticRegressionTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Logistic regression TCP component class.

This class provides methods to compute the value and the gradient of the logistic regression TCP component, as well as to add the logistic regression model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type

object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the logistic regression model.

Type

object of class `LogisticRegressionModel`

parameter_value

Value of the logistic regression model coefficients.

Type

list

intercept_value

Value of the logistic regression model intercept.

Type

list

bounds

See 'Parameters'. Transformed by the inverse sigmoid function.

Type
list

Overview

Table 111: Methods

get_intercept_value()	Get the value of the intercept.
set_intercept_value(*args)	Set the value of the intercept.
add_model()	Add the logistic regression model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

get_intercept_value()

Get the value of the intercept.

Returns
Value of the intercept.

Return type
list

set_intercept_value(**args*)

Set the value of the intercept.

Parameters
***args** (*tuple*) – Keyworded parameters. args[0] should give the value to be set.

add_model()

Add the logistic regression model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**LQPoissonTCP**(*alpha=None*, *beta=None*, *volume_parameter=None*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.RadiobiologyComponentClass*

Linear-quadratic Poisson TCP component class.

This class provides methods to compute the value and the gradient of the linear-quadratic Poisson TCP component.

Parameters

- **alpha** (*int or float, default=None*) – Alpha coefficient for the tumor volume (in the LQ model).
- **beta** (*int or float, default=None*) – Beta coefficient for the tumor volume (in the LQ model).
- **volume_parameter** (*int or float, default=None*) – Dose-volume effect parameter.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 112: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**LymanKutcherBurmanNTCP**(*tolerance_dose_50=None*, *slope_parameter=None*, *volume_parameter=None*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.RadiobiologyComponentClass*

Lyman-Kutcher-Burman (LKB) NTCP component class.

This class provides methods to compute the value and the gradient of the LKB NTCP component.

Parameters

- **tolerance_dose_50** (*int or float, default=None*) – Tolerance value for the dose at 50% tumor control.
- **slope_parameter** (*int or float, default=None*) – Slope parameter.
- **volume_parameter** (*int or float, default=None*) – Dose-volume effect parameter.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 113: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**MaximumDVH**(*target_dose=None*, *quantile_volume=None*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.ConventionalComponentClass*

Maximum dose-volume histogram (Maximum DVH) component class.

This class provides methods to compute the value and the gradient of the maximum DVH component.

Parameters

- **target_dose** (*int or float, default=None*) – Target value for the dose.
- **quantile_volume** (*int or float, default=None*) – Volume level at which to evaluate the dose quantile.

- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 114: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**MeanDose**(*target_dose=None, embedding='active', weight=1.0, bounds=None, link=None, identifier=None, display=True*)

Bases: *pyanno4rt.optimization.components.ConventionalComponentClass*

Mean dose component class.

This class provides methods to compute the value and the gradient of the mean dose component.

Parameters

- **target_dose** (*int or float, default=None*) – Target value for the dose.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 115: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**MinimumDVH**(*target_dose=None*, *quantile_volume=None*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.ConventionalComponentClass*

Minimum dose-volume histogram (Minimum DVH) component class.

This class provides methods to compute the value and the gradient of the minimum DVH component.

Parameters

- **target_dose** (*int or float, default=None*) – Target value for the dose.
- **quantile_volume** (*int or float, default=None*) – Volume level at which to evaluate the dose quantile.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 116: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

`compute_gradient`(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**NaiveBayesNTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Naive Bayes NTCP component class.

This class provides methods to compute the value and the gradient of the naive Bayes NTCP component, as well as to add the naive Bayes model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

`data_model_handler`

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

`model`

The object used to preprocess, tune, train, inspect and evaluate the naive Bayes model.

Type
object of class `NaiveBayesModel`

`parameter_value`

Value of the naive Bayes model parameters.

Type
list

bounds

See 'Parameters'.

Type
list

Overview

Table 117: Methods

add_model()	Add the naive Bayes model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the naive Bayes model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**NaiveBayesTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Naive Bayes TCP component class.

This class provides methods to compute the value and the gradient of the naive Bayes TCP component, as well as to add the naive Bayes model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the naive Bayes model.

Type
object of class `NaiveBayesModel`

parameter_value

Value of the naive Bayes model parameters.

Type
list

bounds

See 'Parameters'.

Type
list

Overview

Table 118: Methods

add_model()	Add the naive Bayes model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the naive Bayes model to the component.

compute_value(**args*)

Compute the component value.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component function.

Return type

float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**NeuralNetworkNTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Neural network NTCP component class.

This class provides methods to compute the value and the gradient of the neural network NTCP component, as well as to add the neural network model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the neural network model.

Type
object of class `NeuralNetworkModel`

parameter_value

Value of the neural network model parameters.

Type
list

bounds

See 'Parameters'. Transformed by the inverse sigmoid function.

Type
list

Overview

Table 119: Methods

add_model()	Add the neural network model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the neural network model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**NeuralNetworkTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Neural network TCP component class.

This class provides methods to compute the value and the gradient of the neural network TCP component, as well as to add the neural network model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the neural network model.

Type
object of class `NeuralNetworkModel`

parameter_value

Value of the neural network model parameters.

Type
list

bounds

See 'Parameters'. Transformed by the inverse sigmoid function.

Type
list

Overview

Table 120: Methods

add_model()	Add the neural network model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the neural network model to the component.

compute_value(**args*)

Compute the component value.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component function.

Return type

float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**RandomForestNTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Random forest NTCP component class.

This class provides methods to compute the value and the gradient of the random forest NTCP component, as well as to add the random forest model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.

- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the random forest model.

Type
object of class `RandomForestModel`

parameter_value

Value of the random forest model parameters.

Type
list

bounds

See 'Parameters'.

Type
list

Overview

Table 121: Methods

add_model()	Add the random forest model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the random forest model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**RandomForestTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Random forest TCP component class.

This class provides methods to compute the value and the gradient of the random forest TCP component, as well as to add the random forest model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type

object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the random forest model.

Type

object of class `RandomForestModel`

parameter_value

Value of the random forest model parameters.

Type

list

bounds

See 'Parameters'.

Type

list

Overview

Table 122: Methods

add_model()	Add the random forest model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the random forest model to the component.

compute_value(**args*)

Compute the component value.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component function.

Return type

float

compute_gradient(**args*)

Compute the component gradient.

Parameters

***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns

Value of the component gradient.

Return type

ndarray

class pyanno4rt.optimization.components.**SquaredDeviation**(*target_dose=None*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.ConventionalComponentClass*

Squared deviation component class.

This class provides methods to compute the value and the gradient of the squared deviation component.

Parameters

- **target_dose** (*int or float, default=None*) – Target value for the dose.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.

- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 123: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**SquaredOverdosing**(*maximum_dose=None, embedding='active', weight=1.0, bounds=None, link=None, identifier=None, display=True*)

Bases: *pyanno4rt.optimization.components.ConventionalComponentClass*

Squared overdosing component class.

This class provides methods to compute the value and the gradient of the squared overdosing component.

Parameters

- **maximum_dose** (*int or float, default=None*) – Maximum value for the dose.

- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 124: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**SquaredUnderdosing**(*minimum_dose=None, embedding='active', weight=1.0, bounds=None, link=None, identifier=None, display=True*)

Bases: *pyanno4rt.optimization.components.ConventionalComponentClass*

Squared underdosing component class.

This class provides methods to compute the value and the gradient of the squared underdosing component.

Parameters

- **minimum_dose** (*int or float, default=None*) – Minimum value for the dose.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

parameter_value

Value of the component parameters.

Type
list

Overview

Table 125: Methods

compute_value(*args)	Return the component value from the jitted 'compute' function.
compute_gradient(*args)	Return the component gradient from the jitted 'differentiate' function.

Members

compute_value(**args*)

Return the component value from the jitted 'compute' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate.

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Return the component gradient from the jitted 'differentiate' function.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**SupportVectorMachineNTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Support vector machine NTCP component class.

This class provides methods to compute the value and the gradient of the support vector machine NTCP component, as well as to add the support vector machine model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the support vector machine model.

Type
object of class `SupportVectorMachineModel`

parameter_value

Value of the primal/dual support vector machine model coefficients.

Type
list

decision_function

Decision function for the fitted kernel type.

Type
callable

decision_gradient

Decision gradient for the fitted kernel type.

Type
callable

bounds

See 'Parameters'. Transformed by the inverse Platt scaling function.

Type
list

Overview

Table 126: Methods

add_model()	Add the support vector machine model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the support vector machine model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

class pyanno4rt.optimization.components.**SupportVectorMachineTCP**(*model_parameters*, *embedding='active'*, *weight=1.0*, *bounds=None*, *link=None*, *identifier=None*, *display=True*)

Bases: *pyanno4rt.optimization.components.MachineLearningComponentClass*

Support vector machine TCP component class.

This class provides methods to compute the value and the gradient of the support vector machine TCP component, as well as to add the support vector machine model.

Parameters

- **model_parameters** (*dict*) – Dictionary with the data handling & learning model parameters, see the class `MachineLearningComponentClass`.
- **embedding** (*{'active', 'passive'}, default='active'*) – Mode of embedding for the component. In 'passive' mode, the component value is computed and tracked, but not considered in the optimization problem, unlike in 'active' mode.
- **weight** (*int or float, default=1.0*) – Weight of the component function.
- **bounds** (*None or list, default=None*) – Constraint bounds for the component.
- **link** (*None or list, default=None*) – Other segments used for joint evaluation.
- **identifier** (*str, default=None*) – Additional string for naming the component.
- **display** (*bool, default=True*) – Indicator for the display of the component.

data_model_handler

The object used to handle the dataset, the feature map generation and the feature (re-)calculation.

Type
object of class `DataModelHandler`

model

The object used to preprocess, tune, train, inspect and evaluate the support vector machine model.

Type
object of class `SupportVectorMachineModel`

parameter_value

Value of the primal/dual support vector machine model coefficients.

Type
list

decision_function

Decision function for the fitted kernel type.

Type
callable

decision_gradient

Decision gradient for the fitted kernel type.

Type
callable

bounds

See 'Parameters'. Transformed by the inverse Platt scaling function.

Type
list

Overview

Table 127: Methods

add_model()	Add the support vector machine model to the component.
compute_value(*args)	Compute the component value.
compute_gradient(*args)	Compute the component gradient.

Members

add_model()

Add the support vector machine model to the component.

compute_value(**args*)

Compute the component value.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component function.

Return type
float

compute_gradient(**args*)

Compute the component gradient.

Parameters
***args** (*tuple*) – Keyworded parameters, where args[0] must be the dose vector(s) to evaluate and args[1] the corresponding segment(s).

Returns
Value of the component gradient.

Return type
ndarray

Attributes

pyanno4rt.optimization.components.**component_map**

pyanno4rt.optimization.initializers

Initializers module.

This module aims to provide methods and classes for initializing the fluence vector by different strategies.

Overview

Table 128: Classes

FluenceInitializer	Fluence initialization class.

Classes

class pyanno4rt.optimization.initializers.**FluenceInitializer**(*initial_strategy*, *initial_fluence_vector*)

Fluence initialization class.

This class provides methods to initialize the fluence vector by different strategies, e.g. towards coverage of the target volumes.

Parameters

- **initial_strategy** (*str*) – Initialization strategy for the fluence vector.
- **initial_fluence_vector** (*list or None*) – User-defined initial fluence vector for the optimization problem.

initial_strategy

See 'Parameters'.

Type
str

initial_fluence_vector

See 'Parameters'.

Type
list or None

Overview

Table 129: Methods

initialize_fluence()	Initialize the fluence vector based on the selected strategy.
initialize_from_data()	Initialize the fluence vector with respect to data medoid points.
initialize_from_target()	Initialize the fluence vector with respect to target coverage.
initialize_from_reference(initial_fluence_ve	Initialize the fluence vector with respect to a reference point.

Members

initialize_fluence()

Initialize the fluence vector based on the selected strategy.

Returns
Initial fluence vector.

Return type
ndarray

initialize_from_data()

Initialize the fluence vector with respect to data medoid points.

Returns
Initial fluence vector.

Return type
ndarray

initialize_from_target()

Initialize the fluence vector with respect to target coverage.

Returns
Initial fluence vector.

Return type
ndarray

initialize_from_reference(*initial_fluence_vector*)

Initialize the fluence vector with respect to a reference point.

Parameters
initial_fluence_vector (*ndarray*) – Reference fluence vector.

Returns
Initial fluence vector.

Return type
ndarray

pyanno4rt.optimization.methods

Optimization methods module.

This module aims to provide different types of optimization methods.

Overview

Table 130: Classes

LexicographicOptimization	Lexicographic problem class.
ParetoOptimization	Pareto problem class.
WeightedSumOptimization	Weighted-sum optimization problem class.

Table 131: Attributes

method_map	-

Classes

class pyanno4rt.optimization.methods.**LexicographicOptimization**(*backprojection*, *objectives*, *constraints*)

Lexicographic problem class.

This class provides methods to implement the lexicographic method for solving the scalarized fluence optimization problem. It features the tracking dictionary and computation functions for the objective, the gradient, and the constraints.

Parameters

- **backprojection** (object of class *DoseProjection* or *ConstantRBEProjection*) – Instance of the class *DoseProjection* or *ConstantRBEProjection*, which inherits from *BackProjection* and provides methods to either compute the dose from the fluence, or the fluence gradient from the dose gradient.
- **objectives** (*tuple*) – Tuple with pairs of segmented structures and their associated objectives.
- **constraints** (*tuple*) – Tuple with pairs of segmented structures and their associated constraints.

backprojection

See 'Parameters'.

Type
object of class *DoseProjection* or *ConstantRBEProjection*

objectives

See 'Parameters'.

Type
tuple

constraints

See 'Parameters'.

Type
tuple

tracker

Dictionary with the objective function values for each iteration, divided by the associated segments.

Type
dict

Overview

Table 132: Attributes

name	-

Table 133: Methods

objective(fluence)	Compute the lexicographic objective function value.
gradient(fluence)	Compute the lexicographic gradient vector.
constraint(fluence)	Compute the lexicographic constraint vector.

Members

name = 'lexicographic'

objective(*fluence*)

Compute the lexicographic objective function value.

Parameters
fluence (*ndarray*) – Values of the fluence.

Returns
total_objective_value – Value of the weighted-sum objective function.

Return type
float

gradient(*fluence*)

Compute the lexicographic gradient vector.

Parameters
fluence (*ndarray*) – Values of the fluence.

Returns
fluence_gradient – Values of the weighted-sum fluence derivatives.

Return type
ndarray

constraint(*fluence*)

Compute the lexicographic constraint vector.

Parameters
fluence (*ndarray*) – Values of the fluence.

Returns
constraints – Values of the constraints.

Return type
ndarray

class pyanno4rt.optimization.methods.**ParetoOptimization**(*backprojection*, *objectives*, *constraints*)

Pareto problem class.

This class provides methods to perform pareto optimization. It implements the respective objective and constraint functions.

Parameters

backprojection (*object of class*)

:param DoseProjection ConstantRBEProjection: The object representing the type of backprojection. :param objectives: Dictionary with the internally configured objectives. :type objectives: dict :param constraints: Dictionary with the internally configured constraints. :type constraints: dict

backprojection

Type

object of class

:class:`~pyanno4rt.optimization.projections._dose_projection.DoseProjection` :class:`~pyanno4rt.optimization.projections._constant_rbe_projection.ConstantRBEProjection`

See 'Parameters'.

objectives

See 'Parameters'.

Type

dict

constraints

See 'Parameters'.

Type

dict

Overview

Table 134: Methods

objective(fluence)	Compute the objective function value(s).
constraint(fluence)	Compute the constraint function value(s).

Members

objective(*fluence*)

Compute the objective function value(s).

Parameters

fluence (*ndarray*) – Fluence vector.

Returns

Objective function value(s).

Return type

list

constraint(*fluence*)

Compute the constraint function value(s).

Parameters

fluence (*ndarray*) – Fluence vector.

Returns

Constraint function value(s).

Return type
list

class pyanno4rt.optimization.methods.**WeightedSumOptimization**(*backprojection*, *objectives*, *constraints*)

Weighted-sum optimization problem class.

This class provides methods to perform weighted-sum optimization. It features a component tracker and implements the respective objective, gradient, constraint and constraint jacobian functions.

Parameters
backprojection (*object of class*)
:param DoseProjection ConstantRBEProjection: The object representing the type of backprojection. :param objectives: Dictionary with the internally configured objectives. :type objectives: dict :param constraints: Dictionary with the internally configured constraints. :type constraints: dict

backprojection

Type
object of class

:class:`~pyanno4rt.optimization.projections._dose_projection.DoseProjection` :class:`~pyanno4rt.optimization.projections._constant_rbe_projection.ConstantRBEProjection`

See 'Parameters'.

objectives

See 'Parameters'.

Type
dict

constraints

See 'Parameters'.

Type
dict

tracker

Dictionary with the iteration-wise component values.

Type
dict

Overview

Table 135: Methods

objective(fluence, track)	Compute the objective function value.
gradient(fluence)	Compute the gradient function value.
constraint(fluence, track)	Compute the constraint function value(s).
jacobian(fluence)	Compute the constraint jacobian function value.

Members

objective(*fluence*, *track=True*)

Compute the objective function value.

Parameters

- **fluence** (*ndarray*) – Fluence vector.
- **track** (*bool, default=True*) – Indicator for tracking the single objective function values.

Returns
Objective function value.

Return type
float

gradient(*fluence*)

Compute the gradient function value.

Parameters
fluence (*ndarray*) – Fluence vector.

Returns
Gradient function value.

Return type
ndarray

constraint(*fluence*, *track=True*)

Compute the constraint function value(s).

Parameters

- **fluence** (*ndarray*) – Fluence vector.
- **track** (*bool, default=True*) – Indicator for tracking the constraint function value(s).

Returns
Constraint function value(s).

Return type
float

jacobian(*fluence*)

Compute the constraint jacobian function value.

Parameters
fluence (*ndarray*) – Fluence vector.

Returns
Constraint jacobian function value.

Return type
ndarray

Attributes

pyanno4rt.optimization.methods.**method_map**

pyanno4rt.optimization.projections

Projections module.

This module aims to provide methods and classes for different types of forward and backward projections between fluence and dose.

Overview

Table 136: Classes

BackProjection	Backprojection superclass.
ConstantRBEProjection	Constant RBE projection class.
DoseProjection	Dose projection class.

Table 137: Attributes

projection_map	-

Classes

class pyanno4rt.optimization.projections.**BackProjection**

Backprojection superclass.

This class provides caching attributes, methods to get/compute the dose and the fluence gradient, and abstract methods to implement projection rules within the inheriting classes.

__dose__

Current (cached) dose vector.

Type
ndarray

__dose_gradient__

Current (cached) dose gradient.

Type
ndarray

__fluence__

Current (cached) fluence vector.

Type
ndarray

__fluence_gradient__

Current (cached) fluence gradient.

Type
ndarray

Overview

Table 138: Methods

compute_dose(fluence)	Compute the dose vector from the fluence and update the cache.
compute_fluence_gradient(dose_gradient)	Compute the fluence gradient from the dose gradient and update the cache.
get_dose()	Get the dose vector.
get_fluence_gradient()	Get the fluence gradient.
compute_dose_result(fluence)	abc Compute the dose projection from the fluence vector.
compute_fluence_gradient_result(dose_	abc Compute the fluence gradient projection from the dose gradient.

Members

compute_dose(*fluence*)

Compute the dose vector from the fluence and update the cache.

Parameters
fluence (*ndarray*) – Values of the fluence vector.

Returns
Values of the dose vector.

Return type
ndarray

compute_fluence_gradient(*dose_gradient*)

Compute the fluence gradient from the dose gradient and update the cache.

Parameters
dose_gradient (*ndarray*) – Values of the dose gradient.

Returns
Values of the fluence gradient.

Return type
ndarray

get_dose()

Get the dose vector.

Returns
Values of the dose vector.

Return type
ndarray

get_fluence_gradient()

Get the fluence gradient.

Returns
Values of the fluence gradient.

Return type
ndarray

abstract compute_dose_result(*fluence*)

Compute the dose projection from the fluence vector.

Parameters
fluence (*ndarray*) – Values of the fluence vector.

Returns
Values of the dose vector.

Return type
ndarray

abstract compute_fluence_gradient_result(*dose_gradient*)

Compute the fluence gradient projection from the dose gradient.

Parameters
dose_gradient (*ndarray*) – Values of the dose gradient.

Returns
Values of the fluence gradient.

Return type
ndarray

class pyanno4rt.optimization.projections.**ConstantRBEProjection**

Bases: *pyanno4rt.optimization.projections.BackProjection*

Constant RBE projection class.

This class provides an implementation of the abstract forward and backward projection methods in `Backprojection` by a linear function with a constant RBE value of 1.1.

Overview

Table 139: Methods

compute_dose_result(fluence)	Compute the dose projection from the fluence vector.
compute_fluence_gradient_result(dose_gra	Compute the fluence gradient projection from the dose gradient.

Members

compute_dose_result(*fluence*)

Compute the dose projection from the fluence vector.

Parameters
fluence (*ndarray*) – Values of the fluence vector.

Returns
Values of the dose vector.

Return type
ndarray

compute_fluence_gradient_result(*dose_gradient*)

Compute the fluence gradient projection from the dose gradient.

Parameters
dose_gradient (*ndarray*) – Values of the dose gradient.

Returns
Values of the fluence gradient.

Return type
ndarray

class pyanno4rt.optimization.projections.**DoseProjection**

Bases: *pyanno4rt.optimization.projections.BackProjection*

Dose projection class.

This class provides an implementation of the abstract forward and backward projection methods in `Backprojection` by a linear function with a neutral RBE value of 1.0.

Overview

Table 140: Methods

compute_dose_result(fluence)	Compute the dose projection from the fluence vector.
compute_fluence_gradient_result(dose_gra	Compute the fluence gradient projection from the dose gradient.

Members

compute_dose_result(*fluence*)

Compute the dose projection from the fluence vector.

Parameters
fluence (*ndarray*) – Values of the fluence vector.

Returns
Values of the dose vector.

Return type
ndarray

compute_fluence_gradient_result(*dose_gradient*)

Compute the fluence gradient projection from the dose gradient.

Parameters
dose_gradient (*ndarray*) – Values of the dose gradient.

Returns
Values of the fluence gradient.

Return type
ndarray

Attributes

pyanno4rt.optimization.projections.**projection_map**

pyanno4rt.optimization.solvers

Solvers module.

This module aims to provide methods and classes for wrapping the local and global solution algorithms from the integrated optimization packages.

Subpackages

pyanno4rt.optimization.solvers.configurations

Solution algorithms module.

This module aims to provide functions to configure the solution algorithms for the optimization packages.

Overview

Table 141: Function

configure_proxmin(problem_instance, lower_variable_bounds, upper_variable_bounds, lower_constraint_bounds, upper_constraint_bounds, algorithm, max_iter, tolerance, callback)	Configure the Proxmin solver.
configure_pyanno4rt(problem_instance, lower_variable_bounds, upper_variable_bounds, lower_constraint_bounds, upper_constraint_bounds, algorithm, max_iter, tolerance, callback)	Configure the pyanno4rt solver.
configure_pymoo(number_of_variables, number_of_objectives, number_of_constraints, problem_instance, lower_variable_bounds, upper_variable_bounds, lower_constraint_bounds, upper_constraint_bounds, algorithm, initial_fluence, max_iter, tolerance)	Configure the Pymoo solver.
configure_pypop7(number_of_variables, problem_instance, lower_variable_bounds, upper_variable_bounds, lower_constraint_bounds, upper_constraint_bounds, algorithm, max_iter, tolerance)	Configure the PyPop7 solver.
configure_scipy(problem_instance, lower_variable_bounds, upper_variable_bounds, lower_constraint_bounds, upper_constraint_bounds, algorithm, max_iter, tolerance, callback)	Configure the SciPy solver.

Functions

`pyanno4rt.optimization.solvers.configurations.`**`configure_proxmin`**(*problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *max_iter*, *tolerance*, *callback*)

Configure the Proxmin solver.

Supported algorithms: ADMM, PGM, SDMM.

Parameters

- **problem_instance** (object of class `LexicographicOptimization` `WeightedSumOptimization`) – The object representing the optimization problem.
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.
- **callback** (*callable*) – Callback function from the class `ProxminSolver`.

Returns

- **fun** (*callable*) – Minimization function from the Proxmin library.

- **arguments** (*dict*) – Dictionary with the function arguments.

`pyanno4rt.optimization.solvers.configurations.`**`configure_pyanno4rt`**(*problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *max_iter*, *tolerance*, *callback*)

Configure the pyanno4rt solver.

Supported algorithms: …

Parameters

- **problem_instance** (object of class `LexicographicOptimization` `WeightedSumOptimization`) – The object representing the optimization problem.
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.
- **callback** (*callable*) – Callback function from the class `Pyanno4rtSolver`.

Returns

- **fun** (*callable*) – Minimization function from the pyanno4rt library.
- **arguments** (*dict*) – Dictionary with the function arguments.

`pyanno4rt.optimization.solvers.configurations.`**`configure_pymoo`**(*number_of_variables*, *number_of_objectives*, *number_of_constraints*, *problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *initial_fluence*, *max_iter*, *tolerance*)

Configure the Pymoo solver.

Supported algorithms: NSGA-3.

Parameters

- **number_of_variables** (*int*) – Number of decision variables.
- **number_of_objectives** (*int*) – Number of objective functions.
- **number_of_constraints** (*int*) – Number of constraint functions.
- **problem_instance** (object of class `ParetoOptimization` The object representing the optimization problem.)

- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **initial_fluence** (*ndarray*) – Initial fluence vector.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.

Returns

- **fun** (*callable*) – Minimization function from the Pymoo library.
- **algorithm_object** (object of class from `pymoo.algorithms`) – The object representing the solution algorithm.
- **problem** (object of class from `pymoo.core.problem`) – The object representing the Pymoo-compatible structure of the multi-objective (Pareto) optimization problem.
- **termination** (object of class from `pymoo.termination`) – The object representing the termination criterion.

`pyanno4rt.optimization.solvers.configurations.`**`configure_pypop7`**`(`*number_of_variables*, *problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *max_iter*, *tolerance*`)`

Configure the PyPop7 solver.

Supported algorithms: LMCMA, LMMAES.

Parameters

- **problem_instance** (object of class `LexicographicOptimization` `WeightedSumOptimization`) – The object representing the optimization problem.
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **initial_fluence** (*ndarray*) – Initial fluence vector.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.

Returns

- **fun** (*object*) – The object representing the optimization algorithm.
- **arguments** (*dict*) – Dictionary with the function arguments.

`pyanno4rt.optimization.solvers.configurations.`**`configure_scipy`**(*problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *max_iter*, *tolerance*, *callback*)

Configure the SciPy solver.

Supported algorithms: L-BFGS-B, TNC, trust-constr.

Parameters

- **problem_instance** (object of class `LexicographicOptimization` `WeightedSumOptimization`) – The object representing the optimization problem.
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.
- **callback** (*callable*) – Callback function from the class `SciPySolver`.

Returns

- **fun** (*callable*) – Minimization function from the SciPy library.
- **arguments** (*dict*) – Dictionary with the function arguments.

pyanno4rt.optimization.solvers.internals

Internal algorithms module.

This module aims to provide algorithms for solving the inverse planning problem.

Overview

Table 142: Classes

ProxminSolver	Proxmin wrapper class.
Pyanno4rtSolver	Internal wrapper class.
PymooSolver	Pymoo wrapper class.
PyPop7Solver	PyPop7 wrapper class.
SciPySolver	SciPy wrapper class.

Table 143: Attributes

solver_map	-

Classes

class pyanno4rt.optimization.solvers.**ProxminSolver**(*number_of_variables*, *number_of_constraints*, *problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *initial_fluence*, *max_iter*, *tolerance*)

Proxmin wrapper class.

This class serves as a wrapper for the proximal optimization algorithms from the Proxmin solver. It takes the problem structure, configures the selected algorithm, and defines the method to run the solver.

Parameters

- **number_of_variables** (*int*) – Number of decision variables.
- **number_of_constraints** (*int*) – Number of constraints.
- **problem_instance** (object of class `LexicographicOptimization` `WeightedSumOptimization`) – The object representing the optimization problem.
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **initial_fluence** (*ndarray*) – Initial fluence vector.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*)
- **value.** (*Precision goal for the objective function*)

fun

Minimization function from the Proxmin library.

Type
callable

arguments

Dictionary with the function arguments.

Type
dict

Overview

Table 144: Methods

callback(X, it, objective)	Log the intermediate results after each iteration.
run(initial_fluence)	Run the Proxmin solver.

Members

callback(*X*, *it*, *objective*)

Log the intermediate results after each iteration.

Parameters

- **X** (*ndarray*) – Optimal point of the current iteration.
- **it** (*int*) – Iteration counter.
- **fun** (*callable*) – Objective value function.

run(*initial_fluence*)

Run the Proxmin solver.

Parameters

initial_fluence (*ndarray*) – Initial fluence vector.

Returns

- *ndarray* – Optimized fluence vector.
- *str* – Description for the cause of termination.

class pyanno4rt.optimization.solvers.**Pyanno4rtSolver**(*number_of_variables*, *number_of_constraints*, *problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *initial_fluence*, *max_iter*, *tolerance*)

Internal wrapper class.

This class serves as a wrapper for the internal optimization algorithms. It takes the problem structure, configures the selected algorithm, and defines the method to run the solver.

Parameters

- **number_of_variables** (*int*) – Number of decision variables.
- **number_of_constraints** (*int*) – Number of constraints.
- **problem_instance** (object of class `LexicographicOptimization` `WeightedSumOptimization`) – The object representing the optimization problem.
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.

- **initial_fluence** (*ndarray*) – Initial fluence vector.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.

fun

Minimization function from the pyanno4rt library.

Type
callable

arguments

Dictionary with the function arguments.

Type
dict

Overview

Table 145: Methods

run(initial_fluence)	Run the pyanno4rt solver.

Members

run(*initial_fluence*)

Run the pyanno4rt solver.

Parameters
initial_fluence (*ndarray*) – Initial fluence vector.

Returns

- *ndarray* – Optimized fluence vector.
- *str* – Description for the cause of termination.

class pyanno4rt.optimization.solvers.**PymooSolver**(*number_of_variables*, *number_of_constraints*, *problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *initial_fluence*, *max_iter*, *tolerance*)

Pymoo wrapper class.

This class serves as a wrapper for the multi-objective (Pareto) optimization algorithms from the Pymoo solver. It takes the problem structure, configures the selected algorithm, and defines the method to run the solver.

Parameters

- **number_of_variables** (*int*) – Number of decision variables.
- **number_of_constraints** (*int*) – Number of constraints.
- **problem_instance** (object of class `ParetoOptimization` The object representing the (Pareto) optimization problem.)
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.

- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **initial_fluence** (*ndarray*) – Initial fluence vector.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.

fun

Minimization function from the Pymoo library.

Type
callable

algorithm_object

The object representing the solution algorithm.

Type
object of class from `pymoo.algorithms`

problem

The object representing the Pymoo-compatible structure of the multi-objective (Pareto) optimization problem.

Type
object of class from `pymoo.core.problem`

termination

The object representing the termination criterion.

Type
object of class from `pymoo.termination`

Overview

Table 146: Methods

run(_)	Run the Pymoo solver.

Members

run(_)

Run the Pymoo solver.

Parameters
initial_fluence (*ndarray*) – Initial fluence vector.

Returns

- *ndarray* – Optimized (Pareto) set of fluence vectors.
- *str* – Description for the cause of termination.

class pyanno4rt.optimization.solvers.**PyPop7Solver**(*number_of_variables*, *number_of_constraints*, *problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *initial_fluence*, *max_iter*, *tolerance*)

PyPop7 wrapper class.

This class serves as a wrapper for the population-based optimization algorithms from the PyPop7 solver. It takes the problem structure, configures the selected algorithm, and defines the method to run the solver.

Parameters

- **number_of_variables** (*int*) – Number of decision variables.
- **number_of_constraints** (*int*) – Number of constraints.
- **problem_instance** (object of class `LexicographicOptimization` `WeightedSumOptimization`) – The object representing the optimization problem.
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **initial_fluence** (*ndarray*) – Initial fluence vector.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.

fun

The object representing the optimization algorithm.

Type
object

arguments

Dictionary with the function arguments.

Type
dict

Overview

Table 147: Methods

run(initial_fluence)	Run the PyPop7 solver.

Members

run(*initial_fluence*)

Run the PyPop7 solver.

Parameters

initial_fluence (*ndarray*) – Initial fluence vector.

Returns

- *ndarray* – Optimized fluence vector.
- *str* – Description for the cause of termination.

class pyanno4rt.optimization.solvers.**SciPySolver**(*number_of_variables*, *number_of_constraints*, *problem_instance*, *lower_variable_bounds*, *upper_variable_bounds*, *lower_constraint_bounds*, *upper_constraint_bounds*, *algorithm*, *initial_fluence*, *max_iter*, *tolerance*)

SciPy wrapper class.

This class serves as a wrapper for the local optimization algorithms from the SciPy solver. It takes the problem structure, configures the selected algorithm, and defines the method to run the solver.

Parameters

- **number_of_variables** (*int*) – Number of decision variables.
- **number_of_constraints** (*int*) – Number of constraints.
- **problem_instance** (object of class `LexicographicOptimization` `WeightedSumOptimization`) – The object representing the optimization problem.
- **lower_variable_bounds** (*list*) – Lower bounds on the decision variables.
- **upper_variable_bounds** (*list*) – Upper bounds on the decision variables.
- **lower_constraint_bounds** (*list*) – Lower bounds on the constraints.
- **upper_constraint_bounds** (*list*) – Upper bounds on the constraints.
- **algorithm** (*str*) – Label for the solution algorithm.
- **initial_fluence** (*ndarray*) – Initial fluence vector.
- **max_iter** (*int*) – Maximum number of iterations.
- **tolerance** (*float*) – Precision goal for the objective function value.

fun

Minimization function from the SciPy library.

Type

callable

arguments

Dictionary with the function arguments.

Type

dict

counter

Counter for the iterations.

Type
int

Overview

Table 148: Methods

callback(intermediate_result)	Log the intermediate results after each iteration.
run(initial_fluence)	Run the SciPy solver.

Members

callback(*intermediate_result*)

Log the intermediate results after each iteration.

Parameters
intermediate_result (*dict*) – Dictionary with the intermediate results of the current iteration.

run(*initial_fluence*)

Run the SciPy solver.

Parameters
initial_fluence (*ndarray*) – Initial fluence vector.

Returns

- *ndarray* – Optimized fluence vector.
- *str* – Description for the cause of termination.

Attributes

pyanno4rt.optimization.solvers.**solver_map**

Overview

Table 149: Classes

FluenceOptimizer	Fluence optimization class.

Classes

class pyanno4rt.optimization.**FluenceOptimizer**(*components*, *method*, *solver*, *algorithm*, *initial_strategy*, *initial_fluence_vector*, *lower_variable_bounds*, *upper_variable_bounds*, *max_iter*, *tolerance*)

Fluence optimization class.

This class provides methods to optimize the fluence vector by solving the inverse planning problem. It preprocesses the configuration inputs, sets up the optimization problem and the solver, and allows to compute both optimized fluence vector and optimized 3D dose cube (CT resolution).

Parameters

- **components** (*dict*) – Optimization components for each segment of interest, i.e., objective functions and constraints.
- **method** (*{'lexicographic', 'pareto', 'weighted-sum'}*) – Single- or multi-criteria optimization method.
- **solver** (*{'proxmin', 'pymoo', 'scipy'}*) – Python package to be used for solving the optimization problem.
- **algorithm** (*str*) – Solution algorithm from the chosen solver.
- **initial_strategy** (*{'data-medoid', 'target-coverage', 'warm-start'}*) – Initialization strategy for the fluence vector.
- **initial_fluence_vector** (*list or None*) – User-defined initial fluence vector for the optimization problem, only used if initial_strategy='warm-start'.
- **lower_variable_bounds** (*int, float, list or None*) – Lower bound(s) on the decision variables.
- **upper_variable_bounds** (*int, float, list or None*) – Upper bound(s) on the decision variables.
- **max_iter** (*int*) – Maximum number of iterations taken for the solver to converge.
- **tolerance** (*float*) – Precision goal for the objective function value.

Overview

Table 150: Methods

remove_overlap(voi)	static Remove overlaps between segments.
resize_segments_to_dose()	static Resize the segments from CT to dose grid.
set_optimization_components(components)	static Set the components of the optimization problem.
adjust_parameters_for_fractionation(co	static Adjust the dose parameters according to the number of fractions.
get_variable_bounds(lower, upper, length)	static Get the lower and upper variable bounds in a compatible form.
get_constraint_bounds(constraints)	static Get the lower and upper constraint bounds in a compatible form.
solve()	Solve the optimization problem.
compute_dose_3d(optimized_fluence)	Compute the 3D dose cube from the optimized fluence vector.

Members

static remove_overlap(*voi*)

Remove overlaps between segments.

Parameters

voi (*tuple*) – Tuple with the labels for the volumes of interest.

static resize_segments_to_dose()

Resize the segments from CT to dose grid.

static set_optimization_components(*components*)

Set the components of the optimization problem.

Parameters

components (*dict*) – Optimization components for each segment of interest, i.e., objectives and constraints, in the raw user format.

Returns

- *dict* – Dictionary with the internally configured objectives.
- *dict* – Dictionary with the internally configured constraints.

static adjust_parameters_for_fractionation(*components*)

Adjust the dose parameters according to the number of fractions.

Parameters

components (*dict*) – Dictionary with the internally configured objectives/constraints.

static get_variable_bounds(*lower*, *upper*, *length*)

Get the lower and upper variable bounds in a compatible form.

Parameters

- **lower** (*int, float, list or None*) – Lower bound(s) on the decision variables.
- **upper** (*int, float, list or None*) – Upper bound(s) on the decision variables.
- **length** (*int*) – Length of the initial fluence vector.

Returns

- *list* – Transformed lower bounds on the decision variables.
- *list* – Transformed upper bounds on the decision variables.

static get_constraint_bounds(*constraints*)

Get the lower and upper constraint bounds in a compatible form.

Parameters

constraints (*dict*) – Dictionary with the internally configured constraints.

Returns

- *list* – Transformed lower bounds on the constraints.
- *list* – Transformed upper bounds on the constraints.

solve()

Solve the optimization problem.

compute_dose_3d(*optimized_fluence*)

Compute the 3D dose cube from the optimized fluence vector.

Parameters

optimized_fluence (*ndarray*) – Optimized fluence vector(s).

Returns

Optimized 3D dose cube (CT resolution).

Return type

ndarray

pyanno4rt.patient

Patient module.

This module aims to provide methods and classes for importing and processing patient data.

Subpackages

pyanno4rt.patient.import_functions

Import functions module.

This module aims to provide import functions to extract computed tomography (CT) and segmentation data from the external data file(s).

Overview

Table 151: Function

generate_ct_from_dcm(data, resolution)	Generate the CT dictionary from a folder with DICOM (.dcm) files.
generate_ct_from_mat(data, resolution)	Generate the CT dictionary from a MATLAB (.mat) file.
generate_ct_from_p(data, resolution)	Generate the CT dictionary from a Python binary (.p) file.
generate_segmentation_from_dcm(data, ct_slices, computed_tomography)	Generate the segmentation dictionary from a folder with DICOM (.dcm) files.
generate_segmentation_from_mat(data, computed_tomography)	Generate the segmentation dictionary from a MATLAB (.mat) file.
generate_segmentation_from_p(data, computed_tomography)	Generate the segmentation dictionary from a Python binary (.p) file.
import_from_dcm(path, resolution)	Import the patient data from a folder with DICOM (.dcm) files.
import_from_mat(path, resolution)	Import the patient data from a MATLAB (.mat) file.
import_from_p(path, resolution)	Import the patient data from a Python (.p) file.
read_data_from_dcm(path)	Read the DICOM data from the path.
read_data_from_mat(path)	Read the MATLAB data from the path.
read_data_from_p(path)	Read the Python data from the path.

Functions

`pyanno4rt.patient.import_functions.`**`generate_ct_from_dcm`**(*data*, *resolution*)

Generate the CT dictionary from a folder with DICOM (.dcm) files.

Parameters

- **data** (*tuple*) – Tuple of `pydicom.dataset.FileDataset` objects with information on the CT slices.
- **resolution** (*None or list*) – Imaging resolution for post-processing interpolation of the CT and segmentation data.

Returns
computed_tomography – Dictionary with information on the CT images.

Return type
dict

Raises
ValueError – If either the grid resolutions, the image positions or the dimensionalities are inconsistent.

`pyanno4rt.patient.import_functions.`**`generate_ct_from_mat`**(*data*, *resolution*)

Generate the CT dictionary from a MATLAB (.mat) file.

Parameters

- **data** (*dict*) – Dictionary with information on the CT slices.
- **resolution** (*None or list*) – Imaging resolution for post-processing interpolation of the CT and segmentation data.

Returns
computed_tomography – Dictionary with information on the CT images.

Return type
dict

`pyanno4rt.patient.import_functions.`**`generate_ct_from_p`**(*data*, *resolution*)

Generate the CT dictionary from a Python binary (.p) file.

Parameters

- **data** (*dict*) – Dictionary with information on the CT slices.
- **resolution** (*None or list*) – Imaging resolution for post-processing interpolation of the CT and segmentation data.

Returns
Dictionary with information on the CT images.

Return type
dict

`pyanno4rt.patient.import_functions.`**`generate_segmentation_from_dcm`**(*data*, *ct_slices*, *computed_tomography*)

Generate the segmentation dictionary from a folder with DICOM (.dcm) files.

Parameters

- **data** (object of class `pydicom.dataset.FileDataset`) – The `pydicom.dataset.FileDataset` object with information on the segmented structures.
- **slices** (*tuple*) – Tuple of `pydicom.dataset.FileDataset` objects with information on the CT slices.
- **computed_tomography** (*dict*) – Dictionary with information on the CT images.

Returns
segmentation – Dictionary with information on the segmented structures.

Return type
dict

Raises
ValueError – If the contour sequence for a segment includes out-of-slice points.

`pyanno4rt.patient.import_functions.`**`generate_segmentation_from_mat`**(*data*, *computed_tomography*)

Generate the segmentation dictionary from a MATLAB (.mat) file.

Parameters

- **data** (*ndarray*) – Array with information on the segmented structures.
- **computed_tomography** (*dict*) – Dictionary with information on the CT images.

Returns

segmentation – Dictionary with information on the segmented structures.

Return type

dict

`pyanno4rt.patient.import_functions.`**`generate_segmentation_from_p`**(*data*, *computed_tomography*)

Generate the segmentation dictionary from a Python binary (.p) file.

Parameters

- **data** (*dict*) – Dictionary with information on the segmented structures.
- **computed_tomography** (*dict*) – Dictionary with information on the CT images.

Returns

Dictionary with information on the segmented structures.

Return type

dict

`pyanno4rt.patient.import_functions.`**`import_from_dcm`**(*path*, *resolution*)

Import the patient data from a folder with DICOM (.dcm) files.

Parameters

- **path** (*str*) – Path to the DICOM folder.
- **resolution** (*None or list*) – Imaging resolution for post-processing interpolation of the CT and segmentation data.

Returns

- *dict* – Dictionary with information on the CT images.
- *dict* – Dictionary with information on the segmented structures.

`pyanno4rt.patient.import_functions.`**`import_from_mat`**(*path*, *resolution*)

Import the patient data from a MATLAB (.mat) file.

Parameters

- **path** (*str*) – Path to the MATLAB file.
- **resolution** (*None or list*) – Imaging resolution for post-processing interpolation of the CT and segmentation data.

Returns

- *dict* – Dictionary with information on the CT images.
- *dict* – Dictionary with information on the segmented structures.

`pyanno4rt.patient.import_functions.`**`import_from_p`**(*path*, *resolution*)

Import the patient data from a Python (.p) file.

Parameters

- **path** (*str*) – Path to the Python file.

- **resolution** (*None or list*) – Imaging resolution for post-processing interpolation of the CT and segmentation data.

Returns

- *dict* – Dictionary with information on the CT images.
- *dict* – Dictionary with information on the segmented structures.

`pyanno4rt.patient.import_functions.read_data_from_dcm(path)`

Read the DICOM data from the path.

Parameters

path (*str*) – Path to the DICOM folder.

Returns

- **computed_tomography_data** (*tuple*) – Tuple of `pydicom.dataset.FileDataset` objects with information on the CT slices.
- **segmentation_data** (object of class `pydicom.dataset.FileDataset`) – The object representation of the segmentation data.

`pyanno4rt.patient.import_functions.read_data_from_mat(path)`

Read the MATLAB data from the path.

Parameters

path (*str*) – Path to the MATLAB file.

Returns

- *dict* – Dictionary with information on the CT slices.
- *ndarray* – Array with information on the segmented structures.

`pyanno4rt.patient.import_functions.read_data_from_p(path)`

Read the Python data from the path.

Parameters

path (*str*) – Path to the Python file.

Returns

- *dict* – Dictionary with information on the CT slices.
- *dict* – Dictionary with information on the segmented structures.

Overview

Table 152: Classes

PatientLoader	Patient loading class.

Classes

class pyanno4rt.patient.**PatientLoader**(*imaging_path*, *target_imaging_resolution*)

Patient loading class.

This class provides methods to load patient data from different input formats and generate the computed tomography (CT) and segmentation dictionaries.

Parameters

- **imaging_path** (*str*) – Path to the CT and segmentation data.
- **target_imaging_resolution** (*None or list*) – Imaging resolution for post-processing interpolation of the CT and segmentation data.

imaging_path

See 'Parameters'.

Type
str

target_imaging_resolution

See 'Parameters'.

Type
None or list

Overview

Table 153: Methods

load()	Load the patient data from the path.

Members

load()

Load the patient data from the path.

pyanno4rt.plan

Plan configuration module.

This module aims to provide methods and classes to generate the plan configuration dictionary.

Overview

Table 154: Classes

PlanGenerator	Plan generation class.

Classes

class pyanno4rt.plan.**PlanGenerator**(*modality*)

Plan generation class.

This class provides methods to generate the plan configuration dictionary for the management and retrieval of plan properties and plan-related parameters.

Parameters

modality (*{'photon', 'proton'}*) – Treatment modality, needs to be consistent with the dose calculation inputs.

modality

See 'Parameters'.

Type

{'photon', 'proton'}

Overview

Table 155: Methods

generate()	Generate the plan configuration dictionary.

Members

generate()

Generate the plan configuration dictionary.

pyanno4rt.tools

Tools module.

This module aims to provide helpful functions that improve code readability.

Overview

Table 156: Function

add_square_brackets(text)	Add square brackets to a string-type text.
apply(function, elements)	Apply a function to each element of an iterable.
arange_with_endpoint(start, stop, step)	Return evenly spaced values within an interval, including the endpoint.
copycat(base_class, path)	Create a copycat from a treatment plan snapshot.
custom_round(number)	Round up a number from 5 as the first decimal place, otherwise round down.
deduplicate(elements)	Convert an iterable to a dictionary with index tuple for each element.
flatten(elements)	Convert a nested iterable to a flat one.
get_all_constraints(segmentation)	Return a tuple with the user-assigned constraints.
get_all_objectives(segmentation)	Return a tuple with the user-assigned objectives.
get_constraint_segments(segmentation)	Get a tuple with the segments associated with the constraints.
get_conventional_objectives(segmentation)	Get a tuple with all set conventional objective functions.
get_conventional_constraints(segmentation)	Get a tuple with all set conventional constraint functions.
get_machine_learning_constraints(segmentation)	Get a tuple with all set machine learning model-based constraint functions.
get_machine_learning_objectives(segmentation)	Get a tuple with all set machine learning model-based objective functions.
get_objective_segments(segmentation)	Get a tuple with the segments associated with the objectives.
get_radiobiology_constraints(segmentation)	Get a tuple with the set radiobiology model-based constraint functions.
get_radiobiology_objectives(segmentation)	Get a tuple with the set radiobiology model-based objective functions.
identity(value, *args)	Return the identity of the first input parameter.
inverse_sigmoid(value, multiplier, summand)	Calculate the inverse sigmoid function value.
load_list_from_file(path)	Load a list of values from a file path.
non_decreasing(array)	Test whether an array is non-decreasing.
non_increasing(array)	Test whether an array is non-increasing.
monotonic(array)	Test whether an array is monotonic.
sigmoid(value, multiplier, summand)	Calculate the sigmoid function value.
snapshot(instance, path, include_patient_data, include_dose_matrix, include_model_data)	Take a snapshot of a treatment plan.
replace_nan(elements, value)	Replace NaN in an iterable by a specific value.

Functions

pyanno4rt.tools.**add_square_brackets**(*text*)

Add square brackets to a string-type text.

Parameters

text (*str*) – Input text to be placed in brackets.

Returns

text – Input text with enclosing square brackets (if non-empty string).

Return type

str

pyanno4rt.tools.**apply**(*function*, *elements*)

Apply a function to each element of an iterable.

Parameters

- **function** (*function*) – Function to be applied.
- **elements** (*iterable*) – Iterable over which to loop.

pyanno4rt.tools.**arange_with_endpoint**(*start*, *stop*, *step*)

Return evenly spaced values within an interval, including the endpoint.

Parameters

- **start** (*int or float*) – Starting point of the interval.
- **stop** (*int or float*) – Stopping point of the interval.
- **step** (*int or float*) – Spacing between points in the interval.

Returns

Array of evenly spaced values.

Return type

ndarray

pyanno4rt.tools.**copycat**(*base_class*, *path*)

Create a copycat from a treatment plan snapshot.

Parameters

- **base_class** (class from *base*) – The base treatment plan class from which to create an instance.
- **path** (*str*) – Directory path of the snapshot.

Returns

The instantiated base treatment plan object.

Return type

object of class from *base*

pyanno4rt.tools.**custom_round**(*number*)

Round up a number from 5 as the first decimal place, otherwise round down.

Parameters

number (*int or float*) – The number to be rounded.

Returns

The rounded number.

Return type

float

`pyanno4rt.tools.deduplicate(elements)`

Convert an iterable to a dictionary with index tuple for each element.

Parameters

elements (*iterable*) – Iterable over which to loop.

Returns

Dictionary with the element-indices pairs.

Return type

dict

`pyanno4rt.tools.flatten(elements)`

Convert a nested iterable to a flat one.

Parameters

elements (*iterable*) – (Nested) iterable to be flattened.

Returns

Generator object with the flattened iterable values.

Return type

generator

`pyanno4rt.tools.get_all_constraints(segmentation)`

Return a tuple with the user-assigned constraints.

Parameters

segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns

Flattened tuple with the user-assigned constraints.

Return type

tuple

`pyanno4rt.tools.get_all_objectives(segmentation)`

Return a tuple with the user-assigned objectives.

Parameters

segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns

Flattened tuple with the user-assigned objectives.

Return type

tuple

`pyanno4rt.tools.get_constraint_segments(segmentation)`

Get a tuple with the segments associated with the constraints.

Parameters

segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns

Flattened tuple with the segments associated with the constraints.

Return type

tuple

`pyanno4rt.tools.get_conventional_objectives(segmentation)`

Get a tuple with all set conventional objective functions.

Parameters

segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns
Flattened tuple with all set conventional objective functions.

Return type
tuple

`pyanno4rt.tools.get_conventional_constraints(segmentation)`

Get a tuple with all set conventional constraint functions.

Parameters
segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns
Flattened tuple with all set conventional constraint functions.

Return type
tuple

`pyanno4rt.tools.get_machine_learning_constraints(segmentation)`

Get a tuple with all set machine learning model-based constraint functions.

Parameters
segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns
Flattened tuple with all set machine learning model-based constraint functions.

Return type
tuple

`pyanno4rt.tools.get_machine_learning_objectives(segmentation)`

Get a tuple with all set machine learning model-based objective functions.

Parameters
segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns
Flattened tuple with all set machine learning model-based objective functions.

Return type
tuple

`pyanno4rt.tools.get_objective_segments(segmentation)`

Get a tuple with the segments associated with the objectives.

Parameters
segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns
Flattened tuple with the segments associated with the objectives.

Return type
tuple

`pyanno4rt.tools.get_radiobiology_constraints(segmentation)`

Get a tuple with the set radiobiology model-based constraint functions.

Parameters
segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns
Flattened tuple with the set radiobiology model-based constraint functions.

Return type
tuple

`pyanno4rt.tools.get_radiobiology_objectives(segmentation)`

Get a tuple with the set radiobiology model-based objective functions.

Parameters

segmentation (*dict*) – Dictionary with information on the segmented structures.

Returns

Flattened tuple with the set radiobiology model-based objective functions.

Return type

tuple

`pyanno4rt.tools.identity(value, *args)`

Return the identity of the first input parameter.

Parameters

- **value** (*arbitrary*) – Value to be returned.
- ***args** (*tuple*) – Tuple with optional (non-keyworded) parameters.

Returns

value – See 'Parameters'.

Return type

arbitrary

`pyanno4rt.tools.inverse_sigmoid(value, multiplier=1, summand=0)`

Calculate the inverse sigmoid function value.

Parameters

- **value** (*int, float, tuple or list*) – Value(s) at which to calculate the inverse sigmoid function.
- **multiplier** (*int or float, default=1*) – Multiplicative coefficient in the linear term.
- **summand** (*int or float, default=0*) – Additive coefficient in the linear term.

Returns

Value(s) of the inverse sigmoid function.

Return type

float or tuple

`pyanno4rt.tools.load_list_from_file(path)`

Load a list of values from a file path.

Parameters

path (*str*) – Path to the list file.

Returns

Loaded list of values.

Return type

list

`pyanno4rt.tools.non_decreasing(array)`

Test whether an array is non-decreasing.

Parameters

array (*ndarray*) – One-dimensional input array.

Returns

Indicator for the non-decrease of the array.

Return type
bool

pyanno4rt.tools.**non_increasing**(*array*)

Test whether an array is non-increasing.

Parameters
array (*ndarray*) – One-dimensional input array.

Returns
Indicator for the non-increase of the array.

Return type
bool

pyanno4rt.tools.**monotonic**(*array*)

Test whether an array is monotonic.

Parameters
array (*ndarray*) – One-dimensional input array.

Returns
Indicator for the monotonicity of the array.

Return type
bool

pyanno4rt.tools.**sigmoid**(*value*, *multiplier=1*, *summand=0*)

Calculate the sigmoid function value.

Parameters

- **value** (*int, float, tuple or list*) – Value(s) at which to calculate the sigmoid function.
- **multiplier** (*int or float, default=1*) – Multiplicative coefficient in the linear term.
- **summand** (*int or float, summand=0*) – Additive coefficient in the linear term.

Returns
Value(s) of the sigmoid function.

Return type
float or tuple

pyanno4rt.tools.**snapshot**(*instance*, *path*, *include_patient_data=False*, *include_dose_matrix=False*, *include_model_data=False*)

Take a snapshot of a treatment plan.

Parameters

- **instance** (object of class from *base*) – The base treatment plan class from which to take a snapshot.
- **path** (*str*) – Directory path for the snapshot (folder).

  Note: If the specified path does not reference an existing folder, one is created automatically.

- **include_patient_data** (*bool, default=False*) – Indicator for the storage of the external patient data, i.e., computed tomography and segmentation data.

- **include_dose_matrix** (*bool, default=False*) – Indicator for the storage of the dose-influence matrix.
- **include_model_data** (*bool, default=False*) – Indicator for the storage of the outcome model-related dataset(s).

Raises
AttributeError – If the treatment plan instance has not been configured yet.

`pyanno4rt.tools.replace_nan(elements, value)`

Replace NaN in an iterable by a specific value.

Parameters

- **elements** (*iterable*) – Iterable over which to loop.
- **value** (*arbitrary*) – Value by which to replace NaNs.

Returns
Generator with the replaced elements.

Return type
generator

pyanno4rt.visualization

Visualization module.

The module aims to provide methods and classes to visualize different aspects of the generated treatment plans, with respect to optimization problem analysis, data-driven model review, and treatment plan evaluation.

Subpackages

pyanno4rt.visualization.visuals

Visual elements module.

The module aims to provide methods and classes to be embedded via clickable buttons in the visualization interface.

Overview

Table 157: Classes

CtDoseSlicingWindowPyQt	CT/Dose slicing window (PyQt) class.
DosimetricsTablePlotterMPL	Dosimetrics table (matplotlib) class.
DVHGraphPlotterMPL	Dose-volume histogram plot (matplotlib) class.
FeatureSelectWindowPyQt	Feature selection window (PyQt) class.
IterGraphPlotterMPL	Iterative objective value plot (Matplotlib) class.
MetricsGraphsPlotterMPL	Data models metrics plot (matplotlib) class.
MetricsTablesPlotterMPL	Data models metrics table (matplotlib) class.
NTCPGraphPlotterMPL	Iterative (N)TCP value plot (matplotlib) class.
PermutationImportancePlotterMPL	Data models permutation importance plot (matplotlib) class.

Classes

class pyanno4rt.visualization.visuals.**CtDoseSlicingWindowPyQt**

Bases: PyQt5.QtWidgets.QMainWindow

CT/Dose slicing window (PyQt) class.

This class provides an interactive plot of the patient's CT/dose slices on the axial, sagittal and coronal axes, including the segment contours, dose level curves, and a scrolling and autoplay functionality.

category

Plot category for assignment to the button groups in the visual interface.

Type
string

name

Attribute name of the classes' instance in the visual interface.

Type
string

label

Label of the plot button in the visual interface.

Type
string

Overview

Table 158: Attributes

category	-
name	-
label	-

Table 159: Methods

view()	Open the full-screen view on the CT/dose slicing window.

Members

category = 'Treatment plan evaluation'

name = 'ct_dose_plotter'

label = 'CT/Dose slice plot'

view()

Open the full-screen view on the CT/dose slicing window.

class pyanno4rt.visualization.visuals.**DosimetricsTablePlotterMPL**

Dosimetrics table (matplotlib) class.

This class provides a table with dosimetric values per segment, e.g. mean dose, dose deviation, min/max dose, DVH parameters and quality indicators.

category

Plot category for assignment to the button groups in the visual interface.

Type
string

name

Attribute name of the classes' instance in the visual interface.

Type
string

label

Label of the plot button in the visual interface.

Type
string

Overview

Table 160: Attributes

category	-
name	-
label	-

Table 161: Methods

view()	Open the full-screen view on the dosimetrics table.

Members

category = 'Treatment plan evaluation'

name = 'dosimetrics_plotter'

label = 'Dosimetric value table'

view()

Open the full-screen view on the dosimetrics table.

class pyanno4rt.visualization.visuals.**DVHGraphPlotterMPL**

Dose-volume histogram plot (matplotlib) class.

This class provides a plot with dose-volume histogram curve per segment.

category

Plot category for assignment to the button groups in the visual interface.

Type
string

name

Attribute name of the classes' instance in the visual interface.

Type
string

label

Label of the plot button in the visual interface.

Type
string

Overview

Table 162: Attributes

category	-
name	-
label	-

Table 163: Methods

view()	Open the full-screen view on the dose-volume histogram.

Members

category = 'Treatment plan evaluation'

name = 'dvh_plotter'

label = 'Dose-volume histogram'

view()

Open the full-screen view on the dose-volume histogram.

class pyanno4rt.visualization.visuals.**FeatureSelectWindowPyQt**

Bases: PyQt5.QtWidgets.QMainWindow

Feature selection window (PyQt) class.

This class provides an interactive plot of the iterative feature values, including a combo box for feature selection, a graph plot with the value per iteration, and a value table as a second representation.

DATA_DEPENDENT

Indicator for the assignment to model-related plots.

Type
bool

category

Plot category for assignment to the button groups in the visual interface.

Type
string

name

Attribute name of the classes' instance in the visual interface.

Type
string

label

Label of the plot button in the visual interface.

Type
string

Overview

Table 164: Attributes

DATA_DEPENDENT	-
category	-
name	-
label	-

Table 165: Methods

view()	Open the full-screen view on the feature selection window.

Members

DATA_DEPENDENT = True

category = 'Optimization problem analysis'

name = 'features_plotter'

label = 'Iterative feature calculation plot'

view()

Open the full-screen view on the feature selection window.

class pyanno4rt.visualization.visuals.**IterGraphPlotterMPL**

Iterative objective value plot (Matplotlib) class.

This class provides a plot with the iterative objective function values.

category

Plot category for assignment to the button groups in the visual interface.

Type
string

name

Attribute name of the classes' instance in the visual interface.

Type
string

label

Label of the plot button in the visual interface.

Type

string

Overview

Table 166: Attributes

category	-
name	-
label	-

Table 167: Methods

view()	Open the full-screen view on the iterative objective value plot.

Members

category = 'Optimization problem analysis'

name = 'iterations_plotter'

label = 'Iterative objective value plot'

view()

Open the full-screen view on the iterative objective value plot.

class pyanno4rt.visualization.visuals.**MetricsGraphsPlotterMPL**

Data models metrics plot (matplotlib) class.

This class provides metrics plots for the different data-dependent models.

DATA_DEPENDENT

Indicator for the assignment to model-related plots.

Type

bool

category

Plot category for assignment to the button groups in the visual interface.

Type

string

name

Attribute name of the classes' instance in the visual interface.

Type

string

label

Label of the plot button in the visual interface.

Type

string

Overview

Table 168: Attributes

DATA_DEPENDENT	-
category	-
name	-
label	-

Table 169: Methods

view()	Open the full-screen view on the metrics plot.

Members

DATA_DEPENDENT = True

category = 'Data-driven model review'

name = 'metrics_graphs_plotter'

label = 'Evaluation metrics graphs'

view()

Open the full-screen view on the metrics plot.

class pyanno4rt.visualization.visuals.**MetricsTablesPlotterMPL**

Data models metrics table (matplotlib) class.

This class provides the metrics table for the different data-dependent models.

DATA_DEPENDENT

Indicator for the assignment to model-related plots.

Type
bool

category

Plot category for assignment to the button groups in the visual interface.

Type
string

name

Attribute name of the classes' instance in the visual interface.

Type
string

label

Label of the plot button in the visual interface.

Type
string

Overview

Table 170: Attributes

DATA_DEPENDENT	-
category	-
name	-
label	-

Table 171: Methods

view()	Open the full-screen view on the metrics table.

Members

DATA_DEPENDENT = True

category = 'Data-driven model review'

name = 'metrics_tables_plotter'

label = 'Evaluation metrics tables'

view()

Open the full-screen view on the metrics table.

class pyanno4rt.visualization.visuals.**NTCPGraphPlotterMPL**

Iterative (N)TCP value plot (matplotlib) class.

This class provides a plot with the iterative (N)TCP values from each outcome prediction model.

category

Plot category for assignment to the button groups in the visual interface.

Type
string

name

Attribute name of the classes' instance in the visual interface.

Type
string

label

Label of the plot button in the visual interface.

Type
string

Overview

Table 172: Attributes

DATA_DEPENDENT	-
category	-
name	-
label	-

Table 173: Methods

view()	Open the full-screen view on the iterative (N)TCP value plot.

Members

DATA_DEPENDENT = True

category = 'Optimization problem analysis'

name = 'ntcp_plotter'

label = 'Iterative (N)TCP value plot'

view()

Open the full-screen view on the iterative (N)TCP value plot.

class pyanno4rt.visualization.visuals.**PermutationImportancePlotterMPL**

Data models permutation importance plot (matplotlib) class.

This class provides permutation importance plots for the different data-dependent models.

DATA_DEPENDENT

Indicator for the assignment to model-related plots.

Type
bool

category

Plot category for assignment to the button groups in the visual interface.

Type
string

name

Attribute name of the classes' instance in the visual interface.

Type
string

label

Label of the plot button in the visual interface.

Type
string

Overview

Table 174: Attributes

DATA_DEPENDENT	-
category	-
name	-
label	-

Table 175: Methods

view()	Open the full-screen view on the permutation importance plot.

Members

DATA_DEPENDENT = True

category = 'Data-driven model review'

name = 'permutation_importance_plotter'

label = 'Permutation importance boxplots'

view()

Open the full-screen view on the permutation importance plot.

Overview

Table 176: Classes

Visualizer	Visualizer class.

Classes

class pyanno4rt.visualization.**Visualizer**(*parent=None*)

Visualizer class.

This class provides methods to build and launch the visual analysis tool, i.e., it initializes the application, creates the main window, provides the window configuration, and runs the application.

application

Instance of the class *SpyderQApplication* for managing control flow and main settings of the visual analysis tool.

Type

object of class *SpyderQApplication*

Overview

Table 177: Methods

launch()	Launch the visual analysis tool.

Members

launch()

Launch the visual analysis tool.

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

I

M

N

O

P

R

S

T

U

V

W