1 ARLO: Automated Reinforcement Learning Optimizer

This library allows modular pipeline creation to solve RL problems. Two ways:

- Create an AutoRLPipeline block: I expect all the pipelines in the automatic block to end with a model generation block.
  - Jointly optimise the pipelines with no Automatic Blocks.
  - Jointly optimise the pipelines with Automatic Blocks.
- Create a RLPipeline block: A pipeline can end with any block.
  - You do not use Automatic Blocks: you perform some DataGeneration, DataPreparation, FeatureEngineering, ModelGeneration. You specify every hyper-parameter.

    This can be used to test new blocks: take a benchmarked pipeline and swap a block with a new block. You see the impact on the pipeline.
  - You do use Automatic Blocks: you can singly optimise a block of the pipeline between FeatureEngineering and ModelGeneration.

You can move from doing automated RL to hand-crafted RL:

- You can decide what to optimise and how
- You can benchmark new blocks
- You can find promising hyper-parameters configurations for new environments, or new RL algorithms. Create the heatmap of the hyper-parameters and see possible trends in the configuration space.
- You can fine-tune already well performing hyper-parameters configurations.
- Use everything implemented in this library with some of your custom blocks simply wrapping your custom block.
- Work on the blocks and don't worry how to piece everything together: focus on what's important. You want to test a new DataPreparation blocks: simply create a Class for it and instantiate a RL pipeline with the blocks already present in the library. Then benchmark your result with a pipeline that does not have your new DataPreparation block.

Units of the library:

- DataGeneration blocks: an environment enters the block and a dataset exits the block. This dataset will be used for learning a RL algorithm.

  These blocks can only be inserted in an Offline RL pipeline.
- DataPreparation blocks: a dataset enters the block and a dataset exits the block. You can impute missing data, shuffle data, augment data.

  These blocks can only be inserted in an Offline RL pipeline.
- FeatureEngineering blocks: a dataset and-or an environment enter the block and a dataset and-or an environment exit the block. You can discretise the state-action spaces, perform reward shaping, wrap the environment, apply transformations to the state-action spaces. Automated hyper-parameter tuning and algorithm selection can be applied.

- ModelGeneration blocks: a dataset and-or an environment enters the block and a policy exits: an RL algorithm is applied. Automated hyper-parameter tuning and algorithm selection can be applied.
- RLPipeline blocks: you can put together various blocks to perform tasks or sub-tasks in RL. You can just have a DataGeneration and DataPreparation blocks, or you can have an entire pipeline with a block from each of the possible Classes of blocks. Automated hyper-parameter tuning and algorithm selection can be applied.
- Tuner blocks: these blocks are used to perform hyper-parameter tuning. These can be used onto FeatureEngineering blocks, ModelGeneration blocks or RLPipeline blocks.

  Specify your wanted Tuner and your wanted Metric and InputLoader.
- Metric blocks: these are used to measure the goodness of a certain block. For example it can compute the DiscountedReward of an agent acting in an environment.
- InputLoader blocks: these are used to decide how to transform the data to pass to each trial in the tuning procedure. For example you may have a selection of environments and you want to train on the randomly.

You can specify any custom unit you want.

Note that the hyper-parameters of all blocks must be specified as *HyperParameter* objects.

This library makes use of the following libraries:

- abc, numpy, scipy, pandas, cloudpickle, joblib, math, copy, os, datetime, itertools, matplotlib
- optuna
- plotly
- torch, gym, xgboost, catboost, MushroomRL, sklearn

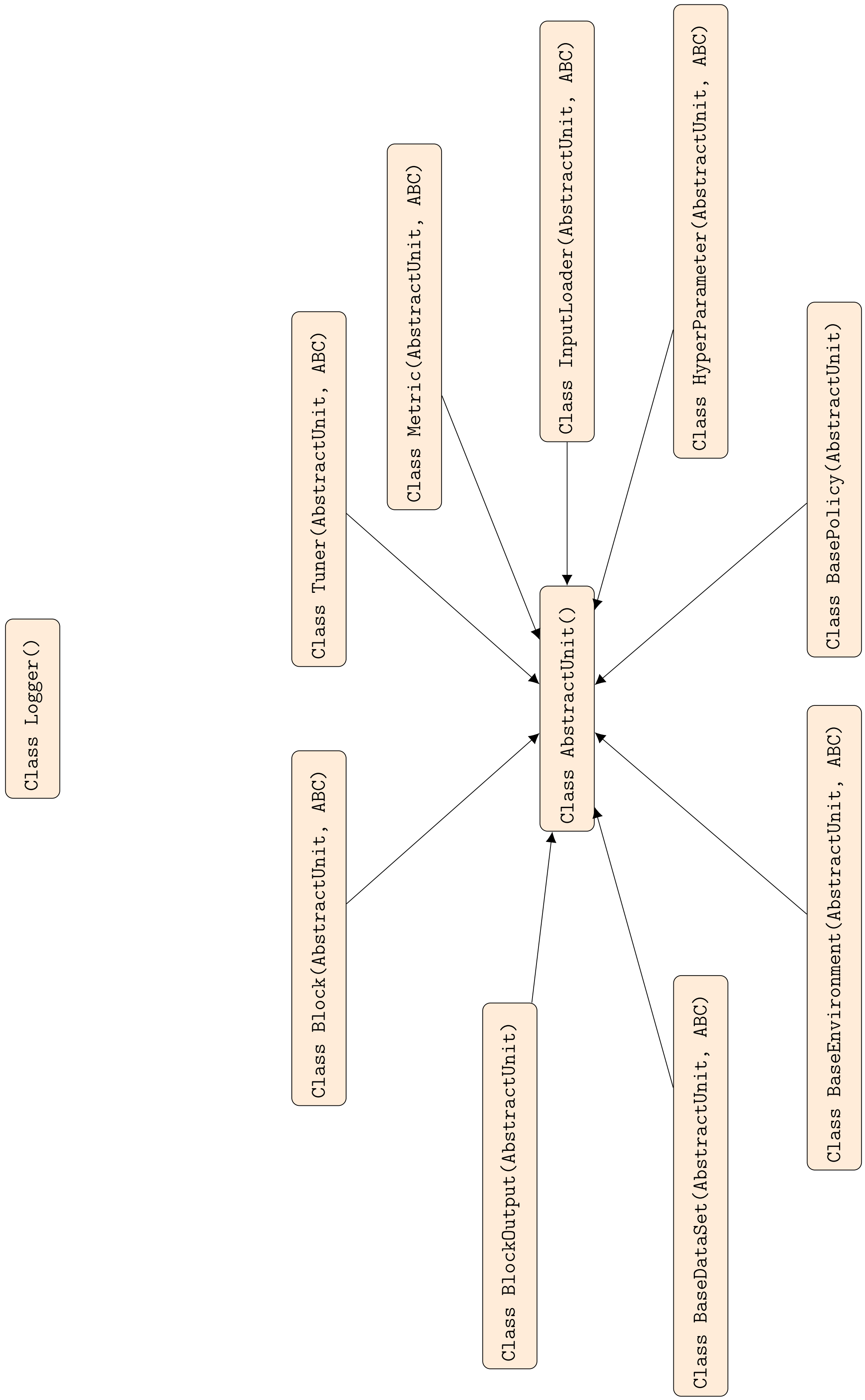
List of implemented Classes for each unit:

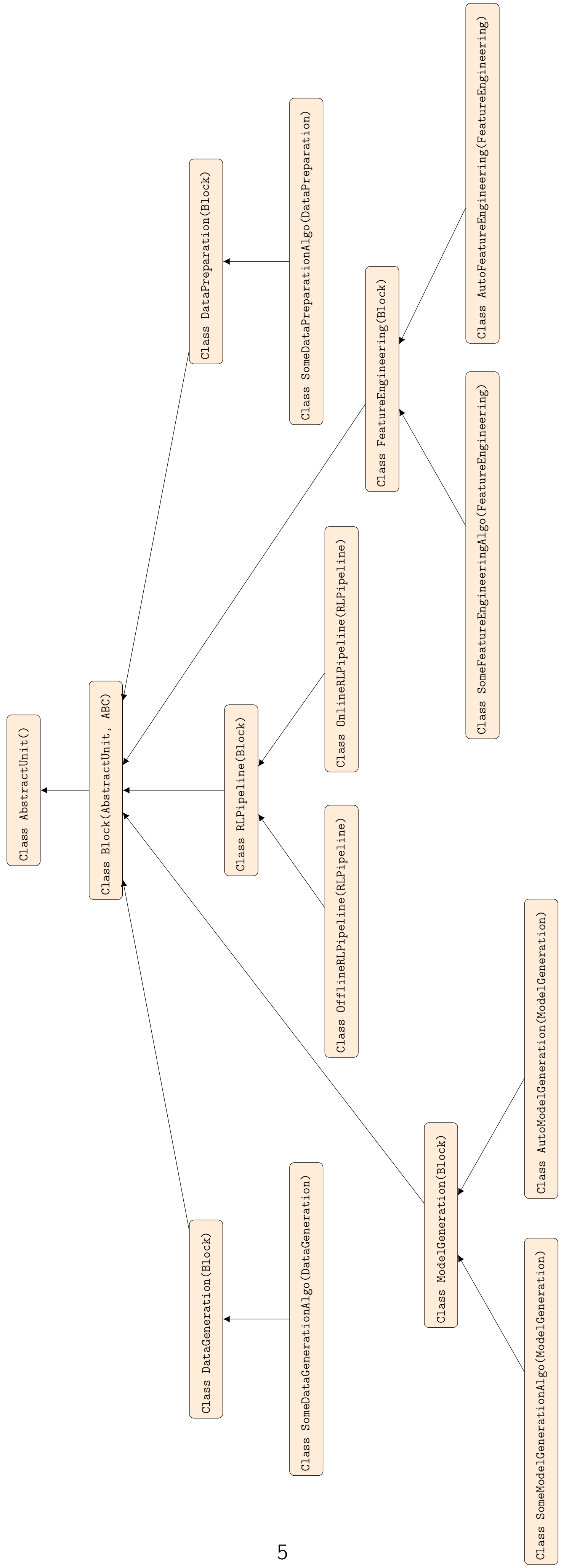
- **DataGeneration:**
  - DataGenerationRandomUniformPolicy
  - DataGenerationMEPOL
- **Data Preparation:**
  - DataPreparationIdentity
  - DataPreparation1NNImputation
  - DataPreparationMeanImputation
- **Feature Engineering:**
  - FeatureEngineeringIdentity
  - FeatureEngineeringRFS
  - FeatureEngineeringFSCMI
  - FeatureEngineeringNystroemMap
- **Model Generation:**
  - ModelGenerationMushroomOnlineDQN

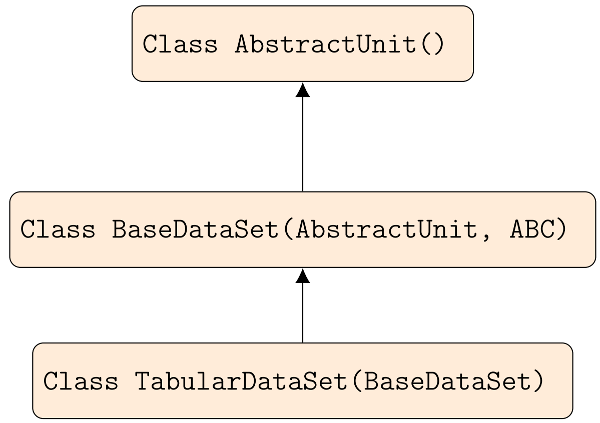
- ModelGenerationMushroomOnlinePPO
- ModelGenerationMushroomOnlineSAC
- ModelGenerationMushroomOnlineDDPG
- ModelGenerationMushroomOnlineGPOMDP
- ModelGenerationMushroomOfflineFQI
- ModelGenerationMushroomOfflineDoubleFQI
- ModelGenerationMushroomOfflineLSPI

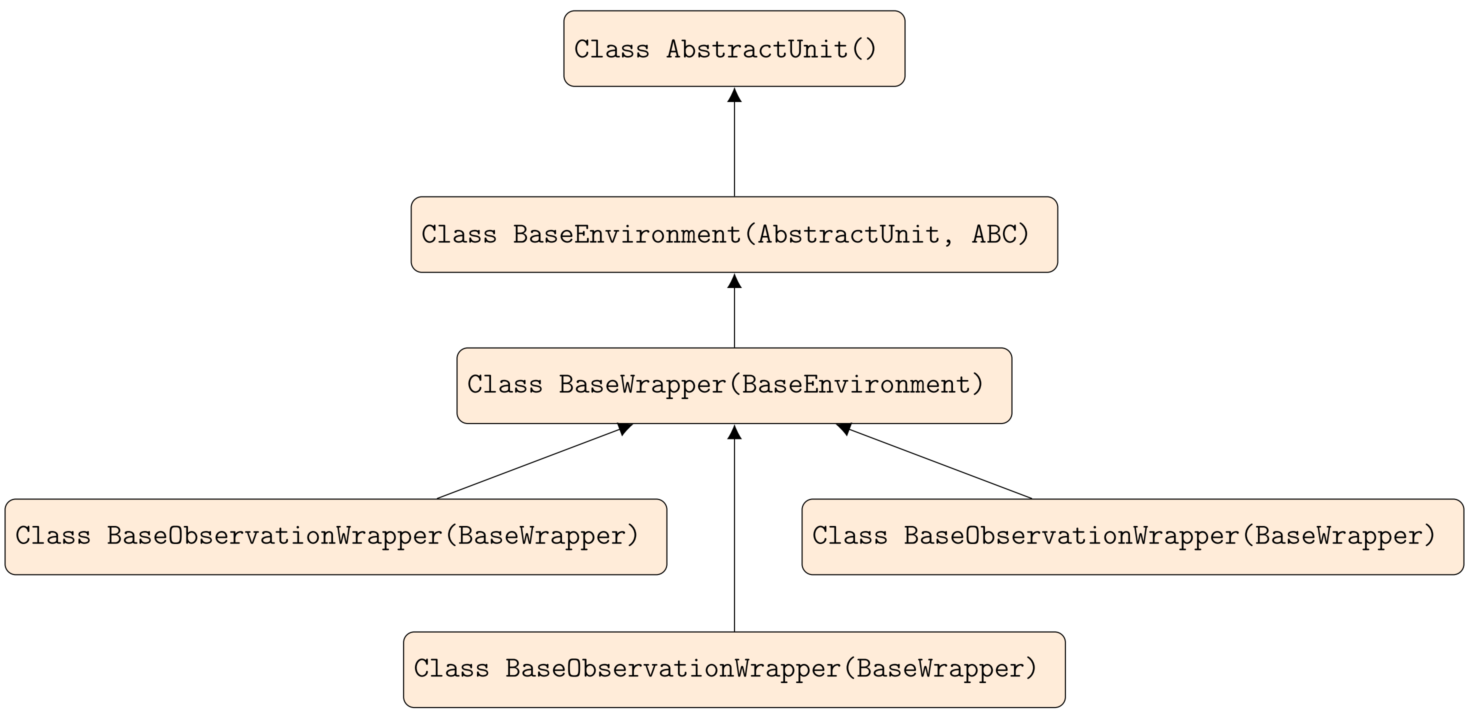
- **Environment:**
  - BaseGridWorld
  - BaseCarOnHill
  - BaseCartPole
  - BaseInvertedPendulum
  - LQG
  - BaseHalfCheetah
  - BaseAnt
  - BaseHopper
  - BaseHumanoid
  - BaseSwimmer
  - BaseWalker2d
- **Input Loader:**
  - LoadSameEnv
  - LoadSameTrainData
  - LoadUniformSubSampleWithReplacement
  - LoadUniformSubSampleWithReplacementAndEnv
  - LoadDifferentSizeForEachBlock
  - LoadDifferentSizeForEachBlockAndEnv
- **Metric:**
  - TDError
  - DiscountedReward
  - TimeSeriesRollingAverageDiscountedReward
  - SomeSpecificMetric
- **Tuner:**
  - TunerOptuna
  - TunerGenetic

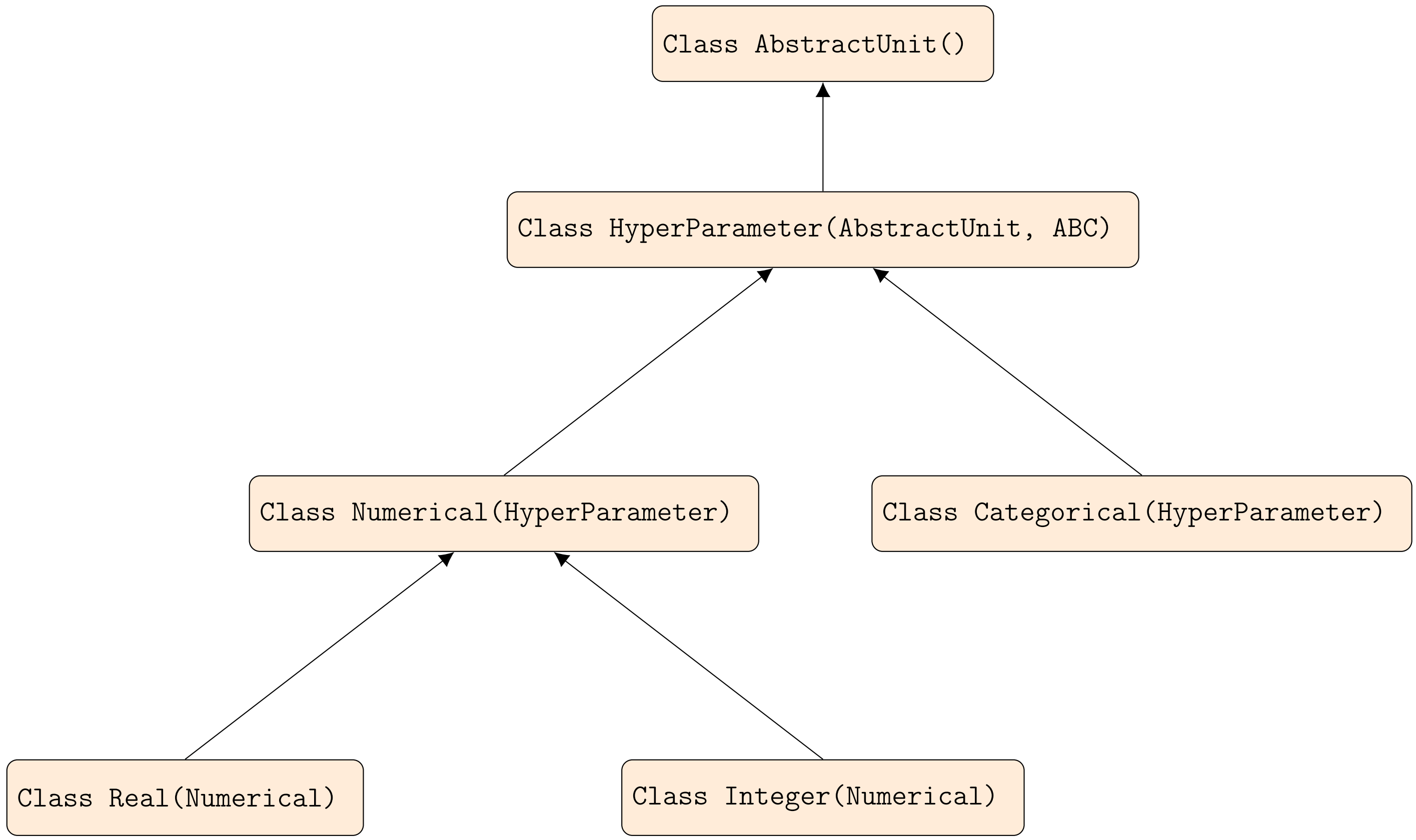
2 High level view of the class structure of the library

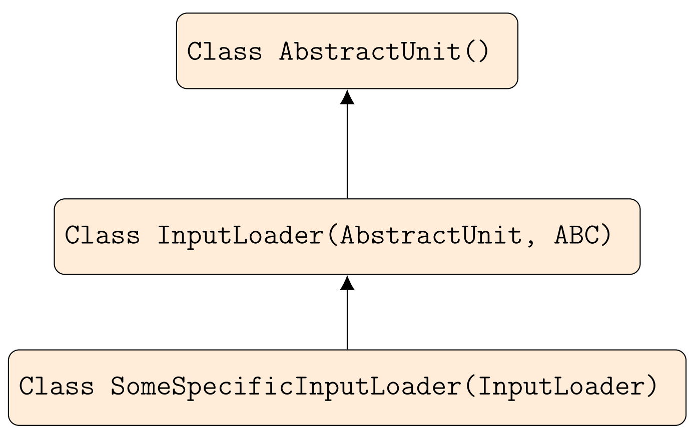


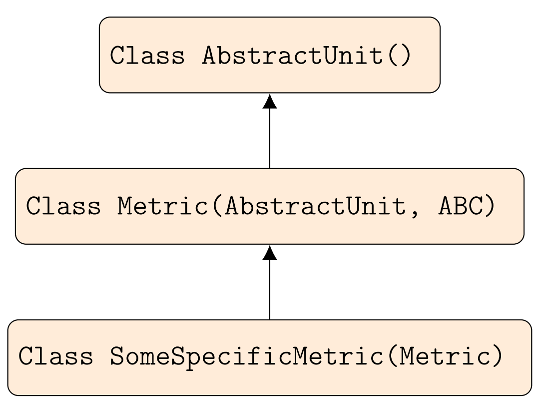

Class AbstractUnit()
Class Block(AbstractUnit, ABC)
Class DataPreparation(Block)
Class SomeDataPreparationAlgo(DataPreparation)
Class FeatureEngineering(Block)
Class AutoFeatureEngineering(FeatureEngineering)
Class SomeFeatureEngineeringAlgo(FeatureEngineering)
Class RLPipeline(Block)
Class OnlineRLPipeline(RLPipeline)
Class OfflineRLPipeline(RLPipeline)
Class ModelGeneration(Block)
Class AutoModelGeneration(ModelGeneration)
Class SomeModelGenerationAlgo(ModelGeneration)
Class DataGeneration(Block)
Class SomeDataGenerationAlgo(DataGeneration)

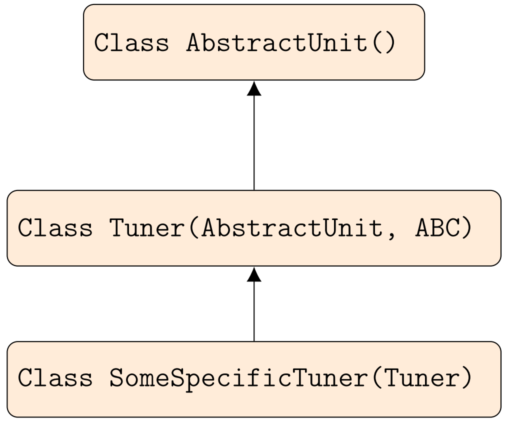

Class AbstractUnit()
Class BaseDataSet(AbstractUnit, ABC)
Class TabularDataSet(BaseDataSet)


Class AbstractUnit()
Class BaseEnvironment(AbstractUnit, ABC)
Class BaseWrapper(BaseEnvironment)
Class BaseObservationWrapper(BaseWrapper)
Class BaseObservationWrapper(BaseWrapper)
Class BaseObservationWrapper(BaseWrapper)


Class AbstractUnit()
Class HyperParameter(AbstractUnit, ABC)
Class Numerical(HyperParameter)
Class Categorical(HyperParameter)
Class Real(Numerical)
Class Integer(Numerical)


Class AbstractUnit()
Class InputLoader(AbstractUnit, ABC)
Class SomeSpecificInputLoader(InputLoader)


Class AbstractUnit()
Class Metric(AbstractUnit, ABC)
Class SomeSpecificMetric(Metric)


Class AbstractUnit()
Class Tuner(AbstractUnit, ABC)
Class SomeSpecificTuner(Tuner)

3 logger.py

In the module *logger.py* the Class *Logger* is implemented.

3.1 Logger

This Class is used in all the library to perform logging. The Class *Logger* is the only one in the library that does not inherit from the Class *AbstractUnit*.

Initialiser:
__init__(name_obj_logging, verbosity=3, mode='console', log_path=None)

Parameters:

- **name_obj_logging** (*str*) - This is the name of the object that is being logged.
- **verbosity** (*int*, *3*) - This is a positive integer representing the verbosity level. There are 5 different levels of verbosity:
  - If verbosity = *0* then only exceptions are logged.
  - If verbosity $\geq$ *1* then also informations are logged.
  - If verbosity $\geq$ *2* then also warnings are logged.
  - If verbosity $\geq$ *3* then also errors are logged.
  - If verbosity $\geq$ *4* then also debug comments are logged.
- **mode** (*str*, 'console') - This is a string and can assume three valid values:
  - If mode = *'console'* then everything is printed to the console
  - If mode = *'file'* then a file is created. The name of the files is given by: *ARLO* plus the current date and time appended at the end of the name, while the extension is *log* (e.g: ARLO_16_39_48__18_12_2021.log).
  - If mode = *'both'* then everything is printed to the console but also saved in the log file.
- **log_path** (*str*, *None*) - This is a string and it must be the path where to save the log file. If not specified the log file will not be saved.

Methods:

- **_to_console(msg, log_level)** - This method prints to console the log with the logging level and with the name of the object that has called the logging. An example of what could be logged is the following:
  [18-12-2021, 16:39:48][INFO, OnlinePipeline]: Learning the online RL pipeline...

  Parameters:

  - **msg** (*str*) - This is a string and it represents the message to be logged to console.
  - **log_level** (*str*) - This is a string representing the log level which can be: *'INFO'*, *'WARNING'*, *'ERROR'*, *'DEBUG'* or '*EXCEPTION'*.
- **_to_file(msg, log_level)** - This method write to file the log with the logging level and with the name of the object that has called the logging. An example of what could be logged is the following:
  [18-12-2021, 16:39:48][INFO, OnlinePipeline]: Learning the online RL pipeline...

  Parameters:

- **msg** (*str*) - This is a string and it represents the message to be logged to file.
- **log_level** (*str*) - This is a string representing the log level which can be: *'INFO'*, *'WARNING'*, *'ERROR'*, *'DEBUG'* or '*EXCEPTION'*.

- **_log(msg, log_level)** - This method, based on the *mode* parameter of the object of Class Logger, either calls the method: **_to_console(msg, log_level)**, **_to_file(msg, log_level)** or both.

  Parameters:

  - **msg** (*str*) - This is a string and it represents the message to be logged, either to console or to file.
  - **log_level** (*str*) - This is a string representing the log level which can be: *'INFO'*, *'WARNING'*, *'ERROR'*, *'DEBUG'* or '*EXCEPTION'*.

- **info(msg)** - This method logs the message with logging level *'INFO'*.

  Parameters:

  - **msg** (*str*) - This is a string and it represents the message to be logged.

- **warning(msg)** - This method logs the message with logging level *'WARNING'*.

  Parameters:

  - **msg** (*str*) - This is a string and it represents the message to be logged.

- **error(msg)** - This method logs the message with logging level *'ERROR'*.

  Parameters:

  - **msg** (*str*) - This is a string and it represents the message to be logged.

- **debug(msg)** - This method logs the message with logging level *'DEBUG'*.

  Parameters:

  - **msg** (*str*) - This is a string and it represents the message to be logged.

- **exception(msg)** - This method logs the message with logging level *'EXCEPTION'*.

  Parameters:

  - **msg** (*str*) - This is a string and it represents the message to be logged.

---

4 abstract_unit.py

In the module *abstract_unit.py* the Class *AbstractUnit* is implemented. Moreover in this same module the **load()** method is present to load a saved object.

4.1 AbstractUnit

Every Class in this library, except for the Class Logger, inherits from this Class. This Class contains some common parameters, the possibility to update the verbosity of an object, the possibility to save an object via *cloudpickle* and the possibility to set a new seeder and creating a new local PRNG.

This is a base Class.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **obj_name** (*str*) - This is a string representing the name of the object. This is used as name of the pickle file where the object will be saved to.
- **seeder** (*int*, 2) - This must be a non-negative integer for seeding: this will be used for setting the state of the local pseudo random number generator (PRNG).
- **log_mode** (*str*, 'console') - This is a string and can assume three valid values:
  - If log_mode = *'console'* then everything is printed to the console
  - If log _mode = *'file'* then a file is created. The name of the files is given by: *ARLO* plus the current date and time appended at the end of the name, while the extension is *log* (e.g: ARLO_16_39_48__18_12_2021.log).
  - If log_mode = *'both'* then everything is printed to the console but also saved in the log file.
- **checkpoint_log_path** (*str*, None) - This is the path where the file containing all logging is saved to. If it is not specified no log file will be saved.

  This is also the path where the object will be saved to. If the path is not specified the object will not be saved. Note that the object will be saved to a pickle file in binary form.
- **verbosity** (*int*, 3) - This is an integer which can be: *0*, *1*, *2*, *3*, *4*. The higher its value the more informations are logged.
- **n_jobs** (*int*, 1) - This is the number of jobs to be used to run some parts of the code of the various Classes in parallel. The jobs can be processes or threads: to pick between these there is the parameter **job_type**.

  It must be a number greater than or equal to *1*, since it may be used for deciding how to parallelise the code, hence the value *-1* is not supported.
- **job_type** (*str*, 'process') - This is a string and it is either *'process'* or *'thread'*: if *'process'* then multiple processes will be created, else if *'thread'* then multiple threads will be created.

  Threads can be used to parallelise some parts of the code that release the GIL (running non-Python code): If the GIL is not released then it is best to use processes:

- cf. https://stackoverflow.com/questions/1912557/a-question-on-python-gil

---

Non-Parameters Members:

- **local_prng** (*numpy.random.default_rng*) - This is the local pseudo random number generator (PRNG). It is an object of Class *numpy.random.default_rng* and it is seeded using the **seeder**.
- **logger** (*Logger*) - This is an object of Class *Logger* that will do the logging.
- **backend** (*str*) - This is a string and it is used in *Joblib* to parallelise the code. It can be: *'multiprocessing'*, *'loky'* or *'threading'*. This is assigned based on the value of the parameter **job_type**.
- **prefer** (*str*) - This is a string and it is used in *Joblib* to parallelise the code. It can be: *'processes'* or *'threads'*. This is assigned based on the value of the parameter **job_type**.

Methods:

- **set_local_prng(new_seeder)** - This method updates the seeder and creates a new local PRNG seeded with the new given seeder.

  Parameters:

  - **new_seeder** (*int*) - This is a non-negative integer and it represents the new seeder to be used for creating a new local PRNG.

    Without this *numpy.random* is not always safe to use on concurrent threads or processes:

    * cf. https://numpy.org/doc/stable/reference/random/parallel.html
    * cf. https://albertcthomas.github.io/good-practices-random-number-generators/

- **save()** - This method saves the object in a pickle file in binary form. The name of the file is equal to the name given to the object that is trying to be saved, plus the current time and date.

  Note that if **checkpoint_log_path** is not specified then no file will be saved.

- **update_verbosity(new_verbosity)** - This method updates the verbosity of this object and the verbosity of the Logger object in the **logger** parameter of this object.

  Parameters:

  - **new_verbosity** (*int*) - This is a positive integer and it represents the new verbosity.

4.2 load() method

load(pickled_file_path) - This method loads a pickled object and returns the loaded object.

Parameters:

- **pickled_file_path** (*str*) - This must be a string containing the absolute path to the pickled file that you want to load.

Returns:

- **loaded_class_obj** (*AbstractUnit*) - This is the loaded object: it will be an object of a Class inheriting from the Class *AbstractUnit*.

---

5 block.py

In the module *block.py* the Class *Block* is implemented.

5.1 Block

This Class is used to represent a generic block: these blocks can make up the pipeline, it can be an automatic block or it can be a pipeline itself. The Class *Block* is an abstract Class, and so it inherits from *ABC*, but it also inherits from the Class *AbstractUnit*.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **eval_metric** (*Metric*) - This is the metric by which the specific block will be ranked. It must be an object of a Class inheriting from the Class *Metric*.

Non-Parameters Members:

- **works_on_online_rl** (*bool*) - This is either *True* or *False*. It is *True* if the block works on online reinforcement learning, while it is *False* otherwise.
- **works_on_offline_rl** (*bool*) - This is either *True* or *False*. It is *True* if the block works on offline reinforcement learning, while it is **False** otherwise.
- **works_on_box_action_space** (*bool*) - This is either *True* or *False*. It is *True* if the block works on box action spaces, while it is *False* otherwise.
- **works_on_discrete_action_space** (*bool*) - This is either *True* or *False*. It is *True* if the block works on discrete action spaces, while it is *False* otherwise.
- **works_on_box_observation_space** (*bool*) - This is either *True* or *False*. It is *True* if the block works on box observation spaces, while it is *False* otherwise.
- **works_on_discrete_observation_space** (*bool*) - This is either *True* or *False*. It is *True* if the block works on discrete observation spaces, while it is *False* otherwise.
- **pipeline_type** (*str*) - This is either *'online'* or *'offline'* and it represents the type of pipeline in which the current block is present. This is needed to know in each block the type of pipeline it is being inserted in.
- **is_learn_successful** (*bool*, False) - This is *True* if the block was learnt properly, *False* otherwise.
- **is_parametrised** (*bool*) - This is either *True* or *False*. It is *True* if the block is parametrised, meaning that it has parameters that can be optimised. One can also create a block with **is_parametrised** equal to *False*: this tells the Class *Tuner* not to do anything with such a block.
- **block_eval** (*float*, None) - This is used by the Class *Tuner* to save the evaluation of the block corresponding to a specific set of hyper parameters with respect to the evaluation metric that is being used in the Class *Tuner*.

  This is useful for having the information about the block evaluation even after the Class Tuner method tune is called (i.e: we can access this information in automatic blocks).

Methods:

- **pre_learn_check(train_data=None, env=None)** - Before learning a block it must be checked that the selected block works on the chosen problem: it must be checked that the block works in offline and-or online reinforcement learning , that it works in continuous and-or discrete, action and-or observation spaces.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - If the block can work with the given offline and-or online reinforcement learning problem and environment spaces this method returns *True*, else it returns *False*.

- **learn(train_data=None, env=None)** - This method checks that the **pipeline_type** of every block is either *'online'* or *'offline'* and that at least one of **train_data** and **env** are not *None* and that they are of the appropriate type.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*BlockOutput*) - This method returns an object of Class *BlockOutput* in one of the three cases below:
    * The **pipeline_type** is neither *'online'* nor *'offline'*.
    * **train_data** and **env** are both *None*.
    * **train_data** is not *None* and **train_data** is not an object of a Class inheriting from the Class *BaseDataSet*, or **env** is not **None** and **env** is not an object of a Class inheriting from the Class *BaseEnvironment*.

    Otherwise this method does not return anything.

- **get_metric()** - This method is used to extract the evaluation metric (or KPI) used by the block to assess the quality of the learnt algorithm present in the block.

  Returns:

  - (*Metric*) - The evaluation metric (or KPI) used by the block to assess the quality of the learnt algorithm present in the block. It is an object of a Class inheriting from the Class *Metric*.

- **set_metric(new_metric)** - This method changes the evaluation metric of the block.

  Parameters:

  - **new_metric** (*Metric*) - This must be an object of a Class inheriting from the Class *Metric*.

  Returns:

  - (*bool*) - This method returns *True* if **new_metric** was set successfully, else it returns *False*.

- **update_verbosity(new_verbosity)** - This method calls the *base* method **update_verbosity()** implemented in the Class *AbstractUnit* that updates the **verbosity** of the block and also the **verbosity** of the *logger* present in the block. Then this method calls the method **update_verbosity()** of the **eval_metric** present in the block.

  Parameters:

- **new_verbosity** (*int*) - This must be a positive integer representing the new **verbosity**.

6 rl_pipeline.py

In the module *rl_pipeline.py* the Class *RLPipeline* is implemented.

6.1 RLPipeline

This Class serves as base Class for the Classes *OnlineRLPipeline* and *OfflineRLPipeline*.

The Class *RLPipeline* inherits from the Class *Block*. This is an *Abstract Class*.

A pipeline is a collection of blocks: it can contain user defined blocks, specific blocks or automatic blocks.

Initialiser:
__init__(list_of_block_objects, eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **list_of_block_objects** (*list*) - This is the list of the block objects that make up the pipeline. Every block in the list must be an object of a Class inheriting from the Class *Block*.

Non-Parameters Members:

- **list_of_block_objects_upon_instantiation** (*list*, None) - This a deep copy of the original value of **list_of_block_objects**.

Methods:

- **pre_learn_check(train_data=None, env=None)** - This method calls the *base* method **pre_learn_check()** implemented in the Class *Block* that checks that the selected block works on the chosen problem: it must be checked that the pipeline works in offline and-or online reinforcement learning, that it works in continuous and-or discrete, action and-or observation spaces.

  Moreover this method sets **list_of_block_objects** equal to **list_of_block_objects_upon_instantiation**. This is needed for re-loading objects.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - If the pipeline can work with the given offline and-or online reinforcement learning problem and environment spaces this method returns *True*, else it returns *False*.

- **get_params()** - This method calls the **get_params()** method of every block in **list_of_block_objects**, that has **is_parametrised** equal to **True**.

  In order to keep track of which parameter is of which block every time we call the **get_params()** method of a block we modify the non-parameter member **block_owner_flag**. In particular every parameter of the block in position *k* in the **list_of_block_objects** will have **block_owner_flag** equal to *k*.

Returns:

- **entire_dict** (*dict*) - This is a flat dictionary containing all the parameters of all the blocks in the **list_of_block_objects** that have **is_parametrised** equal to **True**.

- **set_params(new_params)** - This method calls the **set_params()** method of every block in the **list_of_block_objects** passing to the **set_params()** method of every block the appropriate dictionary of parameters.

  Note that only blocks that have **is_parametrised** equal to **True** are considered.

  To make sure to pass to each block its correct parameters, every call to the **get_params()** method of the pipeline will modify the non-parameter member **block_owner_flag** of the objects of Classes inheriting from the Class *HyperParameter*, to keep track of which parameter is of which block.

  Parameters:

  - **new_params** (*dict*) - This is a flat dictionary containing all the parameters to be used in all the blocks that have **is_parametrised** equal to **True**.

    It must be a dictionary that does not contain any dictionaries (i.e: all parameters must be at the same level).

  Returns:

  - *(bool)* - This is *True* if **new_params** was set successfully, and it is *False* otherwise.

- **consistency_check(train_data=None, env=None)** - This method checks that the pipeline is consistent, namely it checks that:

  - Every block in the **list_of_block_objects** is an object of a Class inheriting from one of the following Classes: *DataGeneration*, *DataPreparation*, *FeatureEngineering*, *ModelGeneration*.
  - The **list_of_block_objects** is not of length zero.
  - There is no more than one *DataGeneration* block, as it would not make sense to have more.
  - There is no more than one *ModelGeneration* block, as it would not make sense to have more.
  - The **list_of_block_objects** is ordered, namely: *DataGeneration* blocks must come before *DataPreparation* blocks, that must come before *FeatureEngineering* blocks, that must come before *ModelGeneration* blocks.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - This is *True* if the pipeline is consistent, else it is *False*.

- **update_verbosity(new_verbosity)** - This method calls the *base* method **update_verbosity()** implemented in the Class *Block* that updates the **verbosity** of the pipeline, the **verbosity** of the *logger* present in the pipeline and the **verbosity** of the **eval_metric** present in the pipeline.

  Then this method calls the **update_verbosity()** method of every block contained in **list_of_block_objects**.

  Parameters:

- **new_verbosity** (*int*) - This must be a positive integer representing the new **verbosity**.

7 online_rl_pipeline.py

In the module *online_rl_pipeline.py* the Class *OnlineRLPipeline* is implemented.

7.1 OnlineRLPipeline

An *OnlineRLPipeline* is used to solve an online reinforcement learning problem.

The Class *OnlineRLPipeline* inherits from the Class *RLPipeline*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*.
- **works_on_offline_rl** is equal to *False*.
- **works_on_box_action_space** is equal to *True*: This means that the blocks inside **list_of_block_objects** can be of any type.
- **works_on_discrete_action_space** is equal to *True*: This means that the blocks inside **list_of_block_objects** can be of any type.
- **works_on_box_observation_space** is equal to *True*: This means that the blocks inside **list_of_block_objects** can be of any type.
- **works_on_discrete_observation_space** is equal to *True*: This means that the blocks inside **list_of_block_objects** can be of any type.
- **pipeline_type** is equal to *'online'*.

  Moreover each block in this pipeline will have **pipeline_type** equal to *'online'*.
- **is_parametrised** is equal to *True*, hence the library expects that at least one of the blocks in the pipeline has **is_parametrised** equal to *True*.

It is however up to the user to make sure to have the proper blocks that can work with the problem at hand: if this is not the case the **pre_learn_check()** method of a block of the wrong type will return *False*, and hence the **pre_learn_check()** method of the pipeline will return *False* and hence no learning can occur: it will not be possible to successfully call the *learn()* method of the pipeline.

Initialiser:
__init__(list_of_block_objects, eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **learn(train_data=None, env=None)** - This method calls the learn method **learn()** implemented in the Class *Block* that checks the **pipeline_type** and that makes sure that not both **train_data** and **env** are *None*.

  Then the method **consistency_check()** is called: If this returns *True* then we cycle through each block present in the **list_of _block_objects** and for each block we call its method **pre_learn_check()** and then we call its method **learn()**.

  For blocks that are objects of a Class inheriting from the Class *ModelGeneration* between the call to the method

pre_learn_check() and the method **learn()**, a third method is called: the method **full_block_instantiation()**.

After having called the method **learn()** of a block if everything was successful we update the local variables **env**, **policy** and **policy_eval**. These will go on to be inserted in an object inheriting from the Class *BlockOutput*, that will be the return value of this method.

Note that **policy_eval** is a dictionary containing the mean of the evaluation and the variance of the evaluation: it follows that the Classes *Metric* that are used with *ModelGeneration* blocks must have two members: **eval_mean** and **eval_var**. These are used in this method to construct the **policy_eval**.

Parameters:

- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- (*BlockOutput*) - If the pipeline was learnt successfully this object contains: **env**, **policy**, **policy_eval**. Else this object will have *None* values for the members: **env**, **policy**, **policy_eval**.

- **consistency_check(train_data=None, env=None)** - This method first calls the method **consistency_check** of the Class *RLPipeline*. Then it makes sure that there are no blocks of a Class inheriting from the Classes *DataGeneration* and *DataPreparation*, as it would not make sense to have these kind of blocks since this is an *online* pipeline.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - This is *True* if the pipeline is consistent, else it is *False*.

- **analyse()** - This method calls the **analyse()** method of every block present in the **list_of_block_objects**. Note that this method can be called only once the **learn()** method of the pipeline has been called.

- **get_info_MDP(latest_env)** - This method given an environment it extracts the member info of such object which contains the observation space, the action space, gamma and the horizon of the environment.

  This is needed for fully instantiating object of a Class inheriting from the Class *ModelGeneration*: indeed these objects undergo a two-step initialisation.

  Parameters:

  - **latest_env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*mushroom_rl.environment.MDPInfo*) - This is an object of Class *mushroom_rl.environment.MDPInfo* which is a Class implemented in MushroomRL in the module *environment.py*. It contains the observation space, the action space, gamma and the horizon of the environment.
    cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/core/environment.py

8 offline_rl_pipeline.py

In the module *offline_rl_pipeline.py* the Class *OfflineRLPipeline* is implemented.

8.1 OfflineRLPipeline

An *OfflineRLPipeline* is used to solve an offline reinforcement learning problem.

The Class *OfflineRLPipeline* inherits from the Class *RLPipeline*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *True*: This means that the blocks inside **list_of_block_objects** can be of any type.
- **works_on_discrete_action_space** is equal to *True*: This means that the blocks inside **list_of_block_objects** can be of any type.
- **works_on_box_observation_space** is equal to *True*: This means that the blocks inside **list_of_block_objects** can be of any type.
- **works_on_discrete_observation_space** is equal to *True*: This means that the blocks inside **list_of_block_objects** can be of any type.
- **pipeline_type** is equal to *'offline'*.

  Moreover each block in this pipeline will have **pipeline_type** equal to *'offline'*.
- **is_parametrised** is equal to *True*, hence the library expects that at least one of the blocks in the pipeline has **is_parametrised** equal to *True*.

It is however up to the user to make sure to have the proper blocks that can work with the problem at hand: if this is not the case the **pre_learn_check()** method of a block of the wrong type will return *False*, and hence the **pre_learn_check()** method of the pipeline will return *False* and hence no learning can occur: it will not be possible to successfully call the *learn()* method of the pipeline.

Initialiser:
__init__(list_of_block_objects, eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **learn(train_data=None, env=None)** - This method calls the learn method **learn()** implemented in the Class *Block* that checks the **pipeline_type** and that makes sure that not both **train_data** and **env** are *None*.

  Then the method **consistency_check()** is called: If this returns *True* then we cycle through each block present in the **list_of _block_objects** and for each block we call its method **pre_learn_check()** and then we call its method **learn()**.

  For blocks that are objects of a Class inheriting from the Class *ModelGeneration* between the call to the method

pre_learn_check() and the method **learn()**, a third method is called: the method **full_block_instantiation()**.

After having called the method **learn()** of a block if everything was successful we update the local variables **train_data**, **env**, **policy** and **policy_eval**. These will go on to be inserted in an object inheriting from the Class *BlockOutput*, that will be the return value of this method.

Note that **policy_eval** is a dictionary containing the mean of the evaluation and the variance of the evaluation: it follows that the Classes *Metric* that are used with *ModelGeneration* blocks must have two members: **eval_mean** and **eval_var**. These are used in this method to construct the **policy_eval**.

Parameters:

- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- *(BlockOutput)* - If the pipeline was learnt successfully this object contains: **train_data**, **env**, **policy**, **policy_eval**. Else this object will have *None* values for the members: **train_data**, **env**, **policy**, **policy_eval**.

- **consistency_check(train_data=None, env=None)** - This method first calls the method **consistency_check** of the Class *RLPipeline*. Then it makes sure that not both **train_data** and **env** are *None*. Moreover:

  - If both the **train_data** and the **env** are not *None* then it checks that there are no blocks of a Class inheriting from the Class *DataGeneration*: since the **train_data** is provided there is no need to extract a dataset from the **env**.
  - If the **train_data** is not *None*, while the **env** is *None* then it checks that there are no blocks of a Class inheriting from the Class *DataGeneration*: since the **train_data** is provided there is no need to extract a dataset.
  - If the **train_data** is *None*, while the **env** is not *None* then it checks that there are is a block of a Class inheriting from the Class *DataGeneration*: since the **train_data** is not provided we have to extract a dataset from the given **env**.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - This is *True* if the pipeline is consistent, else it is *False*.

- **analyse()** - This method calls the **analyse()** method of every block present in the **list_of_block_objects**. Note that this method can be called only once the **learn()** method of the pipeline has been called.

- **get_info_MDP(latest_dataset)** - This method given a dataset it extracts the member info of such object which contains the observation space, the action space, gamma and the horizon of the environment.

  This is needed for fully instantiating object of a Class inheriting from the Class *ModelGeneration*: indeed these objects undergo a two-step initialisation.

  Parameters:

  - **latest_dataset** (*BaseDataSet*) - This must be an object of a Class inheriting from the Class *BaseDataSet*.

  Returns:

– (*mushroom_rl.environment.MDPInfo*) - This is an object of Class *mushroom_rl.environment.MDPInfo* which is a Class implemented in MushroomRL in the module *environment.py*. It contains the observation space, the action space, gamma and the horizon of the environment.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/core/environment.py

---

9 rl_pipeline_automatic.py

In the module *rl_pipeline_automatic.py* the Class *AutoRLPipeline* is implemented.

9.1 AutoRLPipeline

This Class implements automatic reinforcement learning: given a metric and a list of pipelines this block picks the best reinforcement learning pipeline among the possible ones.

The Class *AutoRLPipeline* inherits from the Class *Block*.

Note that in this Class:

- **works_on_online_rl** is equal to *True* if **online_task** is *True*, else it is *False*.
- **works_on_offline_rl** is equal to *True* if **online_task** is *False*, else it is *False*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'online'* if **works_on_online_rl** is equal to *True*, else it is *'offline'*.
- **is_parametrised** is equal to *False*, hence this block cannot go in a *Tuner* object. Indeed it makes no sense to have **is_parametrised** is equal to *True* because this block is already performing joint-optimisation of the pipeline.

Note that the members **works_on_box_action_space**, **works_on_discrete_action_space**, **works_on_box_observation_space** and **works_on_discrete_observation_space** are *True* so that one can specify the right pipelines for the problem at hand.

If the pipelines provided in the **tuner_blocks_dict** are incompatible with the problem at hand then no tuner will be tuned and the method **learn()** of the *AutoRLPipeline* block will fail.

Initialiser:
__init__(eval_metric, obj_name, online_task=False, seeder=2, tuner_blocks_dict=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **tuner_blocks_dict** (*dict*) - This must be a dictionary where the key is a string while the value is an object of a Class inheriting from the Class *Tuner*.
- **online_task** (*bool*) - This must be *True* if the task is an online RL problem, otherwise if the task is an offline RL problem it must be *False*.

Non-Parameters Members:

- **tuner_blocks_dict_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **tuner_blocks_dict**.

Methods:

- **pre_learn_check(train_data=None, env=None)** - This method simply returns *True* and updates the value of **tuner_blocks_dict** to **tuner_blocks_dict_upon_instantiation**. This is needed for re-loading objects.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - This method overrides the one of the *base* Class *Block* and it always returns *True*. Why is this needed? Because there is only one default dictionary for **tuner_blocks_dict** therefore it contains both online and offline pipelines.

    We do not want to stop learning an automatic block only because it contains an online pipeline while the problem is an offline problem: we just want to skip over it.

- **learn(train_data=None, env=None)** - This method calls the learn method **learn()** implemented in the Class *Block* that checks the **pipeline_type** and that makes sure that not both **train_data** and **env** are *None*.

  Then the method **consistency_check()** is called: If this returns *True* then we cycle through each tuner present in the **tuner_blocks_dict** and for each tuner we call its method **tune()**.

  Before calling the method **tune()** of a tuner:

  - We assign the **pipeline_type** to the **block_to_opt** present in the tuner.
  - We call the method **pre_learn_check()** of the **block_to_opt** present in the tuner. If this returns *False* then the currently selected tuner will be skipped.

  Note that we skip over tuners that have input loader and metric not consistent with each other: this is checked by calling the method **is_metric_consistent_with_input_loader()** of the tuner.

  After having called the method **tune()** of a tuner if everything was successful we update the local variables **best_pipeline** and **best_pipeline_eval**.

  If after having gone through each tuner at least one was successfully tuned then the best pipeline found overall will be learnt over the original starting input given to the *AutoRLPipeline* block, and the corresponding object of Class *BlockOutput* is returned.

  Otherwise if no tuner was tuned successfully, out of the ones present in the **tuner_blocks_dict** an empty object of Class *BlockOutput* is returned.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*BlockOutput*) - If this block was learnt successfully this object contains: **train_data**, **env**, **policy**, **policy_eval**. Else this object will have *None* values for the members: **train_data**, **env**, **policy**, **policy_eval**.

- **consistency_check(train_data=None, env=None)** - This method checks that the *AutoRLPipeline* block is consistent, namely it checks that:

  - All the pipelines terminate with a block of the same type: this is important since we cannot compare pipelines (and it makes no sense) that end with a block of different type.
  - All pipelines satisfy the call to their method **consistency_check()**.

  Note that all the tuners present in the **tuner_blocks_dict** must have the same metric as it does not make sense to compare different tuners with different metrics. This is not checked in the library, so it is up to the user to provide sensible metrics.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - This is *True* if the *AutoRLPipeline* block is consistent, else it is *False*.

- **get_params()** - This method returns a deep copy of **tuner_blocks_dict**.

  Returns:

  - (*dict*) - This is a deep copy of **tuner_blocks_dict**.

- **set_params(new_params_dict)** - This method replaces the current **tuner_blocks_dict** and updates **tuner_blocks_dict_upon_instantiation** to **new_params_dict**.

  Parameters:

  - **new_params_dict** (*dict*) - This must be a flat dictionary for the parameters of all pipelines present in this automatic block: it replaces the current **tuner_blocks_dict**.

  Returns:

  - (*bool*) - This is *True* if **new_params_dict** was set successfully, else it is *False*.

- **analyse()** - This method is not yet implemented. It should analyse the learnt *AutoRLPipeline* block, namely it should evaluate it according to the provided **eval_metric**.

- **update_verbosity(new_verbosity)** - This method calls the *base* method **update_verbosity()** implemented in the Class *Block*. Then for all *Tuner* object in the **tuner_blocks_dict** it calls the corresponding method **update_verbosity()**.

  Parameters:

  - **new_verbosity** (int) - This must be a positive integer representing the new **verbosity**.

10 data_generation.py

In the module *data_generation.py* the Classes *DataGeneration*, *DataGenerationRandomUniformPolicy* and *DataGenerationMEPOL* are implemented.

10.1 DataGeneration

This Class is an abstract class and it is used as generic base class for all data generation blocks.

The Class *DataGeneration* inherits from the Class *Block*. This is an *Abstract Class*.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **_get_sample(env, state, episode_steps, discrete_actions, policy=None)** - This method collects a single sample from the environment made of: the current state, the current taken action, the current obtained reward, the next state (reached by taking the current action in the current state), the done (absorbing) flag and the last (episode terminal) flag.

  A sample of this particular kind is sampled because it is what is required by MushroomRL.

  If the **policy** is *None* then this method first samples an action:

  – If the action space is *Discrete* we call the method **integers()** of the **local_prng**, passing to it the number of actions in the action space.

    In this case a *numpy.array* containing the action is added to the sample that will later be returned. This is done as it is required by MushroomRL algorithms. Moreover the *numpy.array* containing the action is also used to interact with the environment, and so it is passed to the method **step()** of the **env**. Therefore the **env**, even if with a discrete action space must expect a *numpy.array* containing the action. To learn more about this see the sub-section **On environments** of the section **Further Details** at the end of this documentation.

  – If the action space is *Box* we call the method **sample_from_box_action_space()** of the **env**.

  Otherwise the the **policy** is not *None* we call the method **draw_action()** of the **policy** to get the action to use in the current state.

  Parameters:

  – **env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  – **state** (*numpy.array*) - This must be a *numpy array* containing the current state.

  – **discrete_actions** (*bool*) - This is *True* if the environment has discrete actions, else it is *False*.

  – **policy** (*None*) - This can be provided if the dataset needs to be extracted using a specified policy different from the random uniform policy. It must be an object of a Class that has the method **draw_action()** accepting as only parameter a **state**.

  Returns:

- *(numpy.array)* - This is a *numpy.array* containing: the current state, the current taken action, the current obtained reward, the next state (reached by taking the current action in the current state), the done (absorbing) flag and the last (episode terminal) flag.

- **_generate_a_dataset(env, n_samples, discrete_actions, policy=None)** - This method extracts a dataset: a collections of samples, by calling the method **_get_sample()** of this block.

  Parameters:

  - **env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.
  - **n_samples** (int) - This must be a positive integer representing the number of samples to extract from the environment.
  - **discrete_actions** (*bool*) - This is *True* if the environment has discrete actions, else it is *False*.
  - **policy** (*None*) - This can be provided if the dataset needs to be extracted using a specified policy different from the random uniform policy. It must be an object of a Class that has the method **draw_action()** accepting as only parameter a **state**.

  Returns:

  - **new_dataset** (*list*) - This is one dataset with a number of samples equal to **n_samples**: it is a *list* where each component is a *numpy.array*.

- **pre_learn_check(train_data=None, env=None)** - This method calls the *base* method **pre_learn_check()** implemented in the Class *Block*.

  Moreover this method sets **algo_params** to **algo_params_upon_instantiation**. This is needed for re-loading objects.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - If the block can work with the given offline and-or online reinforcement learning problem and environment spaces this method returns *True*, else it returns *False*.

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *Block* that checks the **pipeline_type** and that makes sure that not both **train_data** and **env** are *None*.

  Then since a *DataGeneration* block in order to work needs an **env** it is checked that **env** is not *None*: if the **env** is *None* the call to the method **learn()** fails.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*BlockOutput*) - This method returns an empty object of Class *BlockOutput* if the call to the *base* method **learn()** implemented in the Class *Block* was not successful.

    Otherwise it returns the **env**: this is going to be passed onto a specific *DataGeneration* block.

- **get_params()** - This method returns a deep copy of **algo_params**.

  Returns:

  - (*dict*) - This is a dictionary and it corresponds to a deep copy of **algo_params**.

- **set_params(new_params)** - This method changes the value of **algo_params** and of **algo_params_upon_instantiation**, to that of **new_params**.

  Parameters:

  - **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.

  Returns:

  - (*bool*) - This is *True* if **new_params** was set successfully, else it is *False*.

- **analyse()** - This method is not yet implemented. It should evaluate this block according to the provided **eval_metric**.

10.2 DataGenerationRandomUniformPolicy

This Class implements a specific data generation algorithm: it uses a random uniform policy to pick an action with which we probe the environment.

The Class *DataGenerationRandomUniformPolicy* inherits from the Class *DataGeneration*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*: indeed *DataGeneration* blocks are only needed in an offline RL problem.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'offline'*.
- **is_parametrised** is equal to *True*, indeed this block has one parameter that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then 10000 samples will be collected from the environment, thus the following parameters will be used:

  - 'n_samples': 10000, this is the number of samples to extract.

Non-Parameters Members:

- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.

Methods:

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *DataGeneration*. Then an object of Class *TabularDataSet* is created and its **dataset** member is filled by repeatedly calling the method **_generate_a_dataset()**, implemented in the *base* Class *DataGeneration*.

  Since we want to extract a dataset using a random uniform policy we set **policy** equal to *None* when calling the method **_generate_a_dataset()**, implemented in the *base* Class *DataGeneration*.

  This can be done in parallel: multiple environments are created and from each environment a dataset is extracted, and these are then concatenated into a single list.

  Parameters:

- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the dataset generated in the **train_data** member, which is an object of Class *TabularDataSet*.

  If the procedure was not successful this method returns an object of Class *BlockOutput* but all its members are *None*.

10.3 DataGenerationMEPOL

This Class implements a specific data generation algorithm: it implements Task-Agnostic Exploration via Policy Gradient of a Non-Parametric State Entropy Estimate as described in cf. https://arxiv.org/abs/2007.04640

The implementation is a readaptation of the code present in the GitHub repository associated with the above paper, namely cf. https://github.com/muttimirco/mepol

The Class *DataGenerationMEPOL* inherits from the Class *DataGeneration*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*: indeed *DataGeneration* blocks are only needed in an offline RL problem.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *False*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *False*.
- **pipeline_type** is equal to *'offline'*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then 10000 samples will be collected from the environment, thus the following parameters will be used:

  - 'n_samples': 10000, this is the number of samples to extract.
  - 'train_steps': 100
  - 'batch_size': 5000
  - 'create_policy_learning_rate': 0.00025
  - 'state_filter': *None*
  - 'zero_mean_start': 1
  - 'hidden_sizes': $[300, 300]$
  - 'activation': *ReLU*
  - 'log_std_init': -0.5
  - 'eps': $1e-15$
  - 'k': 30
  - 'kl_threshold': 0.1
  - 'max_off_iters': 20

- 'use_backtracking': 1
- 'backtrack_coeff': 2
- 'max_backtrack_try': 10
- 'learning_rate': $1e-3$
- 'num_traj': 20
- 'traj_len': 500
- 'num_epochs': 100
- 'optimizer': *Adam*
- 'full_entropy_traj_scale': 2
- 'full_entropy_k': 4

Non-Parameters Members:

- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.

Methods:

- **_extract_dataset_using_policy(env)** - This method extracts a dataset from the provided **env** using the **policy** learnt in the previous steps of this block (the one obtained by calling the method **_mepol()** of this block).

  This method calls extracts a dataset by repeatedly calling the method **_generate_a_dataset** implemented in the *base* Class *DataGeneration*.

  Since we want to extract a dataset using the learnt policy we set **policy** equal to the one we found, when calling the method **_generate_a_dataset()**, implemented in the *base* Class *DataGeneration*.

  This can be done in parallel: multiple environments are created and from each environment a dataset is extracted, and these are then concatenated into a single list.

  Parameters:

  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **stacked_dataset** (*list*) - This is a dataset in the from that can be used by MushrromRL algorithms, that is: it is a list of list where each list contains a sample from the environment containing the current state, the current action, the reward, the next state, the absorbing state flag and the episode terminal flag.

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *DataGeneration*.

  Then the method **_mepol()** is called: this produces a policy. Then this policy is going to be used to extract a dataset.

  An object of Class *TabularDataSet* is created and its **dataset** member is filled by calling the method **_extract_dataset_using_policy()**.

  Parameters:

- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the dataset generated in the **train_data** member, which is an object of Class *TabularDataSet*.

  If the procedure was not successful this method returns an object of Class *BlockOutput* but all its members are *None*.

To see more details about the meaning of the parameters used in this block and the functioning of the methods see the paper https://arxiv.org/abs/2007.04640 and the code associated with it.

11 data_preparation.py

In the module *data_preparation.py* the Classes *DataPreparation*, *DataPreparationIdentity*, *DataPreparationImputation*, *DataPreparation1NNImputation* and *DataPreparationMeanImputation* are implemented.

11.1 DataPreparation

This Class is an abstract class and it is used as generic base class for all data preparation blocks.

The Class *DataPreparation* inherits from the Class *Block*. This is an *Abstract Class*.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **pre_learn_check(train_data=None, env=None)** - This method calls the *base* method **pre_learn_check()** implemented in the Class *Block*.

  Moreover this method sets **algo_params** to **algo_params_upon_instantiation**. This is needed for re-loading objects.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - If the block can work with the given offline and-or online reinforcement learning problem and environment spaces this method returns *True*, else it returns *False*.

- **learn(train_data=None, env=None)** - This method calls the learn method **learn()** implemented in the Class *Block* that checks the **pipeline_type** and that makes sure that not both **train_data** and **env** are *None*.

  Then since a *DataPreparation* block in order to work needs the **train_data** it is checked that **train_data** is not *None*: if the **train_data** is *None* the call to the method **learn()** fails.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*BlockOutput*) - This method returns an empty object of Class *BlockOutput* if the call to the *base* method **learn()** implemented in the Class *Block* was not successful.

    Otherwise it returns the **train_data**: this is going to be passed onto a specific *DataPreparation* block.

11.2 DataPreparationIdentity

This Class implements a specific data preparation algorithm: it is an identity block meaning that it will not do anything on the input **train_data**. This can be useful when jointly optimising the pipeline.

The Class *DataPreparationIdentity* inherits from the Class *DataPreparation*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*: indeed *DataPreparation* blocks are only needed in an offline RL problem.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'offline'*.
- **is_parametrised** is equal to *False*, indeed this block has no parameters.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *DataPreparation*. Then since this is an identity block the entering **train_data** is inserted, as is, in an object of Class *BlockOutput* which is then returned.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the dataset, as it entered this block, in the **train_data** member, which is an object of Class *BaseDataSet*.

- **get_params()** - This method does nothing since in this block there are no parameters. This method is kept for having a consistent interface across all blocks.

  Returns:

  - (*None*) - This is always *None*: this block has no parameters.

- **set_params(new_params)** - This method does nothing since in this block there are no parameters. This method is kept for having a consistent interface across all blocks.

  Parameters:

  - **new_params** (*dict*) - This must be a dictionary containing all the parameters needed for this block to work.

Returns:

- (*bool*) - This is always *False*: this block has no parameters.

- **analyse()** - This method is not yet implemented. It should evaluate this block according to the provided **eval_metric**.

11.3 DataPreparationImputation

This Class is used to group together common operations for the imputation blocks.

The Class *DataPreparationImputation* inherits from the Class *DataPreparation*. This is an *Abstract Class*.

Initialiser:

__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **_transform_data_for_imputation(train_data)** - This method assumes to have a vector of data that is of the proper length, and some of its members are *numpy.nan*. This method construct a *numpy.ndarray* from the **dataset** member of the **train_data**.

  Parameters:

  - **train_data** (*TabularDataSet*) - This must be an object of a Class inheriting from the Class *TabularDataSet*.

  Returns:

  - **stacked_dataset** (*numpy.ndarray*) - This is a *numpy.ndarray* containing all the samples of the dataset where the features are the current state, the action, the reward and the next state.

- **get_params()** - This method does nothing since in this block there are no parameters. This method is kept for having a consistent interface across all blocks.

  Returns:

  - (*None*) - This is always *None*: this block has no parameters.

- **set_params(new_params)** - This method does nothing since in this block there are no parameters. This method is kept for having a consistent interface across all blocks.

  Parameters:

  - **new_params** (*dict*) - This must be a dictionary containing all the parameters needed for this block to work.

  Returns:

  - (*bool*) - This is always *False*: this block has no parameters.

- **analyse()** - This method is not yet implemented. It should evaluate this block according to the provided **eval_metric**.

11.4 DataPreparation1NNImputation

This Class implements a specific data preparation algorithm: it preforms imputation of the data using 1-NN. Specifically an object of Class *sklearn.imputed.KNNImputer* is created. Note that only non-boolean data of the dataset can be imputed by this method.

The Class *DataPreparation1NNImputation* inherits from the Class *DataPreparationImputation*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*: indeed *DataPreparation* blocks are only needed in an offline RL problem.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'offline'*.
- **is_parametrised** is equal to *False*, indeed this block has no parameters.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **_impute_data(train_data)** - The method assumes that there are missing values and that there are no missing values in the episode terminal flags, nor in the absorbing state flags.

  This method imputes the missing data using an object of Class *sklearn.imputed.KNNImputer* and returns an updated dataset.

  Parameters:

  - **train_data** (*TabularDataSet*) - This must be an object of a Class inheriting from the Class *TabularDataSet*.

  Returns:

  - **new_tabular_dataset** (*TabularDataSet*) - This is a new object of Class TabularDataSet containing the imputed dataset.

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *DataPreparation*. Then the method **_impute_data()** is called and an object of Class *BlockOutput*, containing the imputed dataset, is returned.

  Parameters:

  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

– **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the imputed dataset, in the **train_data** member, which is an object of Class *TabularDataSet*.

11.5 DataPreparationMeanImputation

This Class implements a specific data preparation algorithm: it preforms mean imputation of the data. Note that only non-boolean data of the dataset can be imputed by this method. Moreover this method works only when both the action space and the observation space are continuous (i.e: *Box*).

The Class *DataPreparationMeanImputation* inherits from the Class *DataPreparationImputation*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*: indeed *DataPreparation* blocks are only needed in an offline RL problem.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *False*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *False*.
- **pipeline_type** is equal to *'offline'*.
- **is_parametrised** is equal to *False*, indeed this block has no parameters.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **_impute_data(train_data)** - The method assumes that there are missing values and that there are no missing values in the episode terminal flags, nor in the absorbing state flags.

  This method imputes the missing data using the mean across the relevant feature and returns an updated dataset.

  Parameters:

  - **train_data** (*TabularDataSet*) - This must be an object of a Class inheriting from the Class *TabularDataSet*.

  Returns:

  - **new_tabular_dataset** (*TabularDataSet*) - This is a new object of Class TabularDataSet containing the imputed dataset.

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *DataPreparation*. Then the method **_impute_data()** is called and an object of Class *BlockOutput*, containing the imputed dataset, is returned.

  Parameters:

  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

- **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the imputed dataset, in the **train_data** member, which is an object of Class *TabularDataSet*.

12 feature_engineering.py

In the module *feature_engineering.py* the Classes *FeatureEengineering*, *FeatureEngineeringIdentity*, *FeatureEngineeringRFS*, *FeatureEngineeringFSCMI* and *FeatureEngineeringNystroemMap* are implemented.

12.1 FeatureEengineering

This Class is an abstract class and it is used as generic base class for all feature engineering blocks.

The Class *FeatureEngineering* inherits from the Class *Block*. This is an *Abstract Class*.

Note that feature engineering blocks are both *online* and *offline* blocks: what they return depends on what enters these blocks:

- If only the **env** enters, then only a modified **env** will be returned.
- If only the **train_data** enters, then only a modified **train_data** will be returned.
- If both **env** and **train_data** enters, then both a modified **env** and a modified **train_data** will be returned.

Remark: When an **env** goes into a *FeatureEngineering* block a wrapped **env** will be returned, but there is a problem. Suppose the *FeatureEngineering* blocks applies Principal Components Analysis (PCA): in the standard offline setting we simply fit the PCA onto the dataset and then apply the learnt transformation on the dataset itself.

If we only have an environment however this cannot be done: we could call both fit and transform of the PCA onto a single sample obtained by the **env** but then this would mean that two samples of the **env** could have different principal components and hence we would have different features across different samples of the **env**.

To overcome this, when an **env** enters a *FeatureEngineering* block, first a dataset is sampled from the **env**: on this dataset the *FeatureEngineering* block is fit, and then in the wrapped **env** we simply apply the learnt transformation without re-fitting the *FeatureEngineering* block every time.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **_select_current_actual_value_from_hp_classes(params_objects_dict)** - This method given a dictionary made of objects of a Class inheriting from the Class *HyperParameter* it constructs a dictionary where the keys are the same as the ones in **params_objects_dict** but the values are the value of the member **current_actual_value** of the *HyperParameter* objects in **params_objects_dict**.

  This method is needed in order to pass appropriate values to objects of other libraries which we need to wrap. For example this method is called before creating an object of Class *sklearn.kernel_approximation.Nystroem*.

  Parameters:

  – **params_objects_dict** (*dict*) - This must be a dictionary where the key is a string and the value is an object of a Class inheriting from the Class *HyperParameter*.

  Returns:

- **algo_params_values** (*dict*) - This is a dictionary with the same keys as the input dictionary but as values the value of the member **current_actual_value** of the *HyperParameter* objects in **params_objects_dict**.

- **pre_learn_check(train_data=None, env=None)** - This method calls the *base* method **pre_learn_check()** implemented in the Class *Block*.

  Moreover this method sets **algo_params** to **algo_params_upon_instantiation**. This is needed for re-loading objects.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - If the block can work with the given offline and-or online reinforcement learning problem and environment spaces this method returns *True*, else it returns *False*.

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *Block*. Then it makes sure that:

  - If this block has **pipeline_type** equal to *'online'* then **env** is not *None*.
  - If this block has **pipeline_type** equal to *'offline'* then **train_data** is not *None*.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*BlockOutput*) -This method returns an empty object of Class *BlockOutput* if the call to the *base* method **learn()** implemented in the Class *Block* was not successful.

    Otherwise it returns the **train_data** and the **env**: these are going to be passed onto a specific *FeatureEngineering* block.

- **get_params()** - This method returns a deep copy of **algo_params**.

  Returns:

  - (*dict*) - This is a dictionary and it corresponds to a deep copy of **algo_params**.

- **set_params(new_params)** - This method changes the value of **algo_params** and of **algo_params_upon_instantiation**, to that of **new_params**.

  Parameters:

  - **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.

  Returns:

  - (*bool*) - This is *True* if **new_params** was set successfully, else it is *False*.

12.2 FeatureEengineeringIdentity

This Class implements a specific feature engineering algorithm: it provides the identity block. This simply returns the **env** and the **train_data** as is without changing anything.

The Class *FeatureEengineeringIdentity* inherits from the Class *FeatureEengineering*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*: indeed *FeatureEngineering* blocks are both *online* and *offline* blocks.
- **works_on_offline_rl** is equal to *True*: indeed *FeatureEngineering* blocks are both *online* and *offline* blocks.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'online'* if **works_on_online_rl** is equal to *True*, else it is *False*.
- **is_parametrised** is equal to *False*, indeed this block has no parameters.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **_feature_engineer_env(old_env)** - This method wraps the **env**.

  Parameters:

  - **old_env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **new_env** (*BaseEnvironment*) - This is an object of a Class inheriting from the Class *BaseEnvironment*. In this case since this is an identity block this is equal to **old_env**.

- **_feature_engineer_data(old_data)** - This method modifies the **train_data**.

  Parameters:

  - **old_data** (*BaseDataSet*) - This must be an object of a Class inheriting from the Class *BaseDataSet*.

  Returns:

  - **new_data** (*BaseDataSet*) - This is an object of a Class inheriting from the Class *BaseDataSet*. In this case since this is an identity block this is equal to **old_data**.

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *FeatureEngineering*. Then:

  - If **train_data** is not *None* the method **_feature_engineer_data()** is called: since this is an identity block the entering **train_data** is not changed.

  - If **env** is not *None* the method **_feature_engineer_env()** is called: since this is an identity block the entering **env** is not changed.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the dataset, as it entered this block, in the **train_data** member, which is an object of Class *BaseDataSet*, and the environment, as it entered this block, in the **env** member, which is an object of Class *BaseEnvironment*.

- **get_params()** - This method does nothing since in this block there are no parameters. This method is kept for having a consistent interface across all blocks.

  Returns:

  - (*None*) - This is always *None*: this block has no parameters.

- **set_params(new_params)** - This method does nothing since in this block there are no parameters. This method is kept for having a consistent interface across all blocks.

  Parameters:

  - **new_params** (*dict*) - This must be a dictionary containing all the parameters needed for this block to work.

  Returns:

  - (*bool*) - This is always *False*: this block has no parameters.

- **analyse()** - This method is not yet implemented. It should evaluate this block according to the provided **eval_metric**.

12.3 FeatureEngineeringRFS

This Class implements a specific feature engineering algorithm: it provides a recursive feature selection algorithm for RL.

First an *XGBRegressor* is fitted on a dataset where the target is the reward and the features are the current state and actions. Then a threshold value is picked and only the meaningful features are kept: this is done via the member *feature_importances_* of the *XGBRegressor*.

We continue applying the above procedure recursively taking as target all the next-states. We stop when all the states have been explained, that is when the only variable deemed important for a next state variable is the previous state variable itself.

At the end the list of states that are selected are the ones that appears as important variables at least once in the procedure.

The Class *FeatureEngineeringRFS* inherits from the Class *FeatureEngineering*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*: indeed *FeatureEngineering* blocks are both *online* and *offline* blocks.
- **works_on_offline_rl** is equal to *True*: indeed *FeatureEngineering* blocks are both *online* and *offline* blocks.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.

  Even though this member is *True* if the action space is discrete no feature selection will be applied to the actions. This is done so that this block can be used on Box Observation Space and any type of Action Space.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *False*: this can only be applied to continuous observation spaces.
- **pipeline_type** is equal to *'online'* if **works_on_online_rl** is equal to *True*, else it is *False*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, algo_params=None, data_gen_block_for_env_wrap=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:
  - 'threshold': 0.1
  - 'n_recursions': 100
  - 'feature_selector': XGBRegressor()
- **data_gen_block_for_env_wrap** (*DataGeneration*, None) - This must be an object of a Class inheriting from the Class *DataGeneration*. This is used to extract a dataset from the environment and this dataset will be used for fitting the feature engineering algorithm.

This must extract an object of a Class inheriting from the Class *TabularDataSet*.

Note that this is only used if the method **learn()** receives an **env** but not the **train_data**.

If this is not provided an object of Class *DataGenerationRandomUniformPolicy* collecting 100000 samples will be used.

To see more in detail why this is needed see the **Remark** before the **Initialiser** of the Class *FeatureEngineering*.

Non-Parameters Members:

- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **count** (*int*, 0) - This is needed to make sure to stop the recursive feature elimination after the prescribed number.

  This is a non-negative integer: after the method **_recursive_call()** has been called more than **n_recursion** times we stop the feature selection procedure.

Methods:

- **_from_tabular_dataset_extract_y(original_train_data, idx_target)** - This method loops over the dataset contained in an object of a Class inheriting from the Class *TabularDataSet* and it extracts the element at index **idx_target**.

  Parameters:

  - **original_train_data** (*TabularDataSet*) - This is the original **train_data** and it must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **idx_target** (**list**) - This is a list containing the index of the target variable.

  Returns:

  - **y** (*numpy.array*) - This is a *numpy.array* containing the values of the target variable. This is the *y* parameter used in the method **fit()** of **XGBRegressor**.

- **_recursive_call(original_train_data, list_of_selected_state_features, list_of_selected_action_features, list_of_states_not_explained)**
  - This method is recursive:

  - Each call to this method increments **count** by one.
  - We exit the method if we are done: either when **count** reaches **n_recursions** or when **list_of_states_not_explained** is empty.
  - For each call to this method:
    - We extract the target from the **train_data** using the method **_from_tabular_dataset_extract_y()**, then we fit the **feature_selector**.
    - From the **feature_selector** we extract the **feature_importances_** and by using the **threshold** we select the meaningful state and action variables.
    - If there is only one meaningful state selected and it coincides with the target then we end, otherwise we recursively call this method on the remaining selected meaningful states.

  Parameters:

- **original_train_data** (*TabularDataSet*) - This is the original **train_data** and it must be an object of a Class inheriting from the Class *TabularDataSet*.
- **list_of_selected_state_features** (*list*) - This is the list containing all the state features that appeared at least once throughout all the recursive calls of this method.
- **list_of_selected_action_features** (*list*) - This is the list containing all the action features that appeared at least once throughout all the recursive calls of this method.
- **list_of_states_not_explained** (*list*) - This is the list used to determine when to stop: it contains the states variables that are deemed not explained: those for which the target next state variable is not explained just by its previous value but by also other state variables.

- **_recursive_feature_selection(original_train_data)** - This method calls the first step of the feature selection procedure: it uses the reward as target and it uses all the current states and action as features.

  Once the important state are selected the method **_recursive_call()** is called on the renaming states.

  If no states are selected the **list_of_selected_state_features** will be *None*, and if no actions are selected the **list_of_selected_action_features** will be *None*.

  Parameters:

  - **original_train_data** (*TabularDataSet*) - This is the original **train_data** and it must be an object of a Class inheriting from the Class *TabularDataSet*.

  Returns:

  - **list_of_selected_state_features** (*list*) - This is the list containing all the state features that have been deemed important. It can be *None*.
  - **list_of_selected_action_features** (*list*) - This is the list containing all the action features that have been deemed important. It can be *None*.

- **_feature_engineer_env(old_env)** - This method wraps the **env**, specifically:
  - A **TabularDataSet** is extracted using the **data_gen_block_for_env_wrap**.
  - Then the method **_recursive_feature_selection()** is called onto the extracted dataset. We now have the list of selected features. Then:
    * If **list_of_selected_state_features** is not *None* a Class *Wrapper* inheriting from the Class *BaseObservationWrapper* is created: the observation space is modified by selecting only the important states present in **list_of_selected_state_features**.

      Moreover the method **observation()** is **overridden**: we call the old **observation()** method and then we select only the important states present in **list_of_selected_state_features**.

      An object, **new_env**, of Class *Wrapper* is created, passing to the initialiser the **old_env**.

      Furthermore if the **action_space** is continuous and if **list_of_selected_action_features** is not *None* a Class *Wrapper* inheriting from the Class *BaseActionWrapper* is created: the action space is modified by selecting only the important actions present in **list_of_selected_action_features**.

      Moreover the method **action()** is **overridden**: we call the old **action()** method and then we select only the important actions present in **list_of_selected_action_features**.

      An object, **new_env**, of Class *Wrapper* is created, passing to the initialiser the **old_env**.

    - If **list_of_selected_state_features** is *None* we return the original **old_env**: the feature selection procedure failed.

  Parameters:

  - **old_env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **new_env** (*BaseEnvironment*) - This is an object of a Class inheriting from the Class *BaseEnvironment*. It is the wrapped **env**.

- **_feature_engineer_data(old_data)** - This method modifies the **train_data**, specifically:
  - The method **_recursive_feature_selection()** is called onto the **old_data**. Then:
    - If **list_of_selected_state_features** is not *None* the **observation_space**, the current states, and the next states present in **old_data** are modified by selecting only the important states present in **list_of_selected_**

      Moreover if the **action_space** is continuous and if **list_of_selected_action_features** is not *None* the **action_space** and the current actions are modified by selecting only the important actions present in **list_of_selected_action_features**.
    - If **list_of_selected_state_features** is *None* we return the original **old_data**: the feature selection procedure failed.

  Parameters:

  - **old_data** (*TabularDataSet*) - This must be an object of a Class inheriting from the Class *TabularDataSet*.

  Returns:

  - **new_data** (*TabularDataSet*) - This is an object of a Class inheriting from the Class *TabularDataSet*. It is the modified **train_data**.

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *FeatureEngineering*. Then:
  - If **train_data** is not *None* and if the **observation_space** is not one dimensional the method **_feature_engineer_data()** is called.

    Otherwise if the **observation_space** is one dimensional the original **train_data** is returned as is.
  - If **env** is not *None* and if the **observation_space** is not one dimensional the method **_feature_engineer_env()** is called.

    Otherwise if the **observation_space** is one dimensional the original **env** is returned as is.

  Parameters:

  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the new dataset in the **train_data** member, which is an object of Class *BaseDataSet*, and the new environment in the **env** member, which is an object of Class *BaseEnvironment*.

- **analyse()** - This method is not yet implemented. It should evaluate this block according to the provided **eval_metric**.

12.4 FeatureEngineeringFSCMI

This Class implements a specific feature engineering algorithm: it performs forward feature selection of the observation space using as metric the mutual information as proposed in Feature Selection via Mutual Information: New Theoretical Insights.

cf.https://arxiv.org/abs/1907.07384

The implementation of this block is based on the implementation associated with the cited paper.

This can be applied only to continuous spaces: it has no effect on discrete spaces.

The Class *FeatureEngineeringFSCMI* inherits from the Class *FeatureEngineering*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*: indeed *FeatureEngineering* blocks are both *online* and *offline* blocks.
- **works_on_offline_rl** is equal to *True*: indeed *FeatureEngineering* blocks are both *online* and *offline* blocks.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.

  Even though this member is *True* if the action space is discrete no feature selection will be applied to the actions. This is done so that this block can be used on Box Observation Space and any type of Action Space.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *False*: this can only be applied to continuous observation spaces.
- **pipeline_type** is equal to *'online'* if **works_on_online_rl** is equal to *True*, else it is *False*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, algo_params=None, data_gen_block_for_env_wrap=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:
  - 'threshold': 0.1
  - 'k': 5
- **data_gen_block_for_env_wrap** (*DataGeneration*, None) - This must be an object of a Class inheriting from the Class *DataGeneration*. This is used to extract a dataset from the environment and this dataset will be used for fitting the feature engineering algorithm.

  This must extract an object of a Class inheriting from the Class *TabularDataSet*.

  Note that this is only used if the method **learn()** receives an **env** but not the **train_data**.

If this is not provided an object of Class *DataGenerationRandomUniformPolicy* collecting 100000 samples will be used.

To see more in detail why this is needed see the **Remark** before the **Initialiser** of the Class *FeatureEngineering*.

Non-Parameters Members:

- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.

Methods:

- **_mixed_mutual_info_forward_fs(features, target)** - This method computes an ordering of the features via Forward Feature Selection, by ranking the features based on the Mutual Information. We discard all the features that have a Mutual Information below the provided *threshold*.

  Parameters:

  - **features** (*numpy.ndarray*) - This is a multi-dimensional *numpy.ndarray* representing the features: this is the current state.
  - **target** (*numpy.ndarray*) - This is a multi-dimensional *numpy.ndarray* representing the target: this is the next state and the reward.

  Returns:

  - **sorted_ids** (*list*) - This is a list of non-negative integers and it corresponds to the selected ordering of the state features.

- **_feature_engineer_env(old_env)** - This method wraps the **env**, specifically:

  - A **TabularDataSet** is extracted using the **data_gen_block_for_env_wrap**.
  - Then the method **_mixed_mutual_info_forward_fs()** is called onto the extracted dataset. We now have the list of selected state features. Then:
    - If **list_of_selected_state_features** is not *None* a Class *Wrapper* inheriting from the Class *BaseObservationWrapper* is created: the observation space is modified by selecting only the important states present in **list_of_selected_state_features**.

      Moreover the method **observation()** is **overridden**: we call the old **observation()** method and then we select only the important states present in **list_of_selected_state_features**.

      An object, **new_env**, of Class *Wrapper* is created, passing to the initialiser the **old_env**.
    - If **list_of_selected_state_features** is *None* we return the original **old_env**: the feature selection procedure failed.

  Parameters:

  - **old_env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **new_env** (*BaseEnvironment*) - This is an object of a Class inheriting from the Class *BaseEnvironment*. It is the wrapped **env**.

- **_feature_engineer_data(old_data)** - This method modifies the **train_data**, specifically:

- The method **_mixed_mutual_info_forward_fs()** is called onto the **old_data**. Then:
  * If **list_of_selected_state_features** is not *None* the **observation_space**, the current states, and the next states present in **old_data** are modified by selecting only the important states present in **list_of_selected_**
  * If **list_of_selected_state_features** is *None* we return the original **old_data**: the feature selection procedure failed.

Parameters:

- **old_data** (*TabularDataSet*) - This must be an object of a Class inheriting from the Class *TabularDataSet*.

Returns:

- **new_data** (*TabularDataSet*) - This is an object of a Class inheriting from the Class *TabularDataSet*. It is the modified **train_data**.

- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *FeatureEngineering*. Then:

  - If **train_data** is not *None* and if the **observation_space** is not one dimensional the method **_feature_engineer_data()** is called.

    Otherwise if the **observation_space** is one dimensional the original **train_data** is returned as is.
  - If **env** is not *None* and if the **observation_space** is not one dimensional the method **_feature_engineer_env()** is called.

    Otherwise if the **observation_space** is one dimensional the original **env** is returned as is.

  Parameters:

  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the new dataset in the **train_data** member, which is an object of Class *BaseDataSet*, and the new environment in the **env** member, which is an object of Class *BaseEnvironment*.

- **analyse()** - This method is not yet implemented. It should evaluate this block according to the provided **eval_metric**.

To see more details about the meaning of the parameters used in this block and the functioning of the methods see the paper https://arxiv.org/abs/1907.07384 and the code associated with it.

12.5 FeatureEngineeringNystroemMap

This Class implements a specific feature engineering algorithm: it constructs an approximate feature map for an arbitrary kernel using a subset of the data as basis.
cf. https://scikit-learn.org/stable/modules/generated/sklearn.kernel_approximation.Nystroem.html

The Class *FeatureEngineeringNystroemMap* inherits from the Class *FeatureEngineering*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*: indeed *FeatureEngineering* blocks are both *online* and *offline* blocks.
- **works_on_offline_rl** is equal to *True*: indeed *FeatureEngineering* blocks are both *online* and *offline* blocks.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *False*: this can only be applied to continuous observation spaces.
- **pipeline_type** is equal to *'online'* if **works_on_online_rl** is equal to *True*, else it is *False*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, algo_params=None, data_gen_block_for_env_wrap=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then a Radial Basis Function Kernel is going to be used with a value of gamma equal to 0.1 and 100 components will be created, using as the members **seeder** and **n_jobs** of this block to specify the random state and the number of jobs to use for parallelism, thus the following parameters will be used:

  - 'kernel': 'rbf'
  - 'gamma': 0.1
  - 'n_components': 100
  - 'random_state': **seeder**
  - 'n_of_jobs': **n_jobs**

- **data_gen_block_for_env_wrap** (*DataGeneration*, None) - This must be an object of a Class inheriting from the Class *DataGeneration*. This is used to extract a dataset from the environment and this dataset will be used for fitting the feature engineering algorithm.

  This must extract an object of a Class inheriting from the Class *TabularDataSet*.

  Note that this is only used if the method **learn()** receives an **env** but not the **train_data**.

  If this is not provided an object of Class *DataGenerationRandomUniformPolicy* collecting 100000 samples will be used.

  To see more in detail why this is needed see the **Remark** before the **Initialiser** of the Class *FeatureEngineering*.

Non-Parameters Members:

- **algo_object** (None) - This is the object containing the actual feature engineering algorithm. In this case it is an object of Class *sklearn.kernel_approximation.Nystroem*.
- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.

Methods:

- **_feature_engineer_env(old_env)** - This method wraps the **env**, specifically:
  - A Class *Wrapper* inheriting from the Class *BaseObservationWrapper* is created: the observation space is modified by calling the **transform** method of the **algo_object** (an object of Class *sklearn.kernel_approximation.Nystroem* in this case).

    Moreover the method **observation()** is **overridden**: we call the old **observation()** method and then we apply the **transform** method of the **algo_object** (an object of Class *sklearn.kernel_approximation.Nystroem* in this case).
  - An object, **new_env**, of Class *Wrapper* is created, passing to the initialiser the **old_env**.

  Parameters:

  - **old_env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **new_env** (*BaseEnvironment*) - This is an object of a Class inheriting from the Class *BaseEnvironment*. It is the wrapped **env**.
- **_feature_engineer_data(old_data)** - This method modifies the **train_data**, specifically:
  - The **transform** method of the **algo_object** (an object of Class *sklearn.kernel_approximation.Nystroem* in this case) is called onto the current states, and onto the next states present in **old_data**.
  - A new object of Class *TabularDataSet* is created with the transformed data, and states.

  Parameters:

  - **old_data** (*TabularDataSet*) - This must be an object of a Class inheriting from the Class *TabularDataSet*.

  Returns:

  - **new_data** (*TabularDataSet*) - This is an object of a Class inheriting from the Class *TabularDataSet*. It is the modified **train_data**.
- **learn(train_data=None, env=None)** - This method calls the *base* method **learn()** implemented in the Class *FeatureEngineering*. Then:
  - If **train_data** is not *None* the method **_feature_engineer_data()** is called.
  - If **env** is not *None* then the **data_gen_block_for_env_wrap** is learnt to extract a dataset from the **env**.

    Then the **fit** method of the **algo_object** (an object of Class *sklearn.kernel_approximation.Nystroem* in this case) is called onto the extracted dataset.

    Finally the method **_feature_engineer_env()** is called.

Parameters:

- **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- **res** (*BlockOutput*) - This is an object of Class *BlockOutput* which contains the new dataset in the **train_data** member, which is an object of Class *BaseDataSet*, and the new environment in the **env** member, which is an object of Class *BaseEnvironment*.

- **analyse()** - This method is not yet implemented. It should evaluate this block according to the provided **eval_metric**.

---

13 model_generation.py

In the module *model_generation.py* the Class *ModelGeneration* is implemented.

13.1 ModelGeneration

This Class used to group all Classes that do ModelGeneration: this includes also AutoModelGeneration.

The Class *ModelGeneration* inherits from the Class *Block*. This is an *Abstract Class*.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **construct_policy(policy, regressor_type, approximator=None)** - This method constructs an object of Class *BasePolicy* from the provided parameters.

  Parameters:

  – **policy** (*mushroom_rl.policy.Policy*) - This is the **policy** used in the RL algorithm implemented in the current *ModelGeneration* block.

    Note that the **policy** alternatively can also be an object of any Class that exposes the method **draw_action()** taking as parameter just a single state.

  – **regressor_type** (*str*) - This is a string and it represents the regressor used in the policy, and it is either: *'action_regressor'*, *'q_regressor'* or *'generic_regressor'*.

  – **approximator** (*mushroom_rl.approximators.regressor.Regressor*, None) - This represents the **approximator** used in the **policy** in the RL algorithm implemented in the current *ModelGeneration* block.

    This is optional since all the relevant information are contained already in the **policy**, however this is used to discriminate between different policies and it plays a role in which *Metric* can be applied to which **policy**. To learn more about this read the documentation about the Class *DiscountedRewardMetric*.

    Note that the **approximator** alternatively can also be an object of any Class that exposes the method **predict()** taking as parameter either a single sample, or multiple samples.

  Returns:

  – (*BasePolicy*) - This is an object of Class *BasePolicy* in which the policy and the approximator are the ones give as parameters to this method.

- **_walk_dict_to_select_current_actual_value(dict_of_hyperparams)** - This method is a recursive method that visits every sub-dictionary in the main dictionary with the aim to create a single flat dictionary from the original one. In doing so it also extracts the member **current_actual_value** from the *HyperParamter* object.

  Parameters:

  – **dict_of_hyperparams** (*dict*) - This is a dictionary that can contain several sub-dictionary, where each value of each dictionary is an object of a Class inheriting from the Class *HyperParameter*.

  Returns:

- **dict_of_hyperparams** (*dict*) - This is a flat dictionary where each value in the dictionary is a value: the value of the member **current_actual _value** of the corresponding *HyperParameter* object.

- **_select_current_actual_value_from_hp_classes(params_structured_dict)** - This method flattens the original dictionary, that can contain multiple sub-dictionaries, and it extracts the member **current_actual_value** from the starting dictionary, that contains *HyperParameter* objects.

  This is achieved by calling the method **_walk_dict_to_select_current_actual_value**.

  Parameters:

  - **params_structured_dict** (*dict*) - This is a dictionary which can contain several sub-dictionaries.

  Returns:

  - **algo_params_values** (*dict*) - This is a flat dictionary that contains the value of the member **current_actual_value** extracted from the original dictionary that contained *HyperParameter* objects.

- **pre_learn_check(train_data=None, env=None)** - This method resets to *False* the member **fully_instantiated**: this means that the block has not been fully instantiated. Then:

  - It calls the *base* method **pre_learn_check()** implemented in the Class *Block*.
  - This method sets **algo_params** to **algo_params_upon_instantiation**. This is needed for re-loading objects.
  - It checks that the **regressor_type** is either one of: *'action_regressor'*, *'q_regressor'* or *'generic_regressor'*. These are the three types of regressor supported by MushroomRL for RL.
    cf. https://mushroomrl.readthedocs.io/en/latest/source/tutorials/tutorials.2_approximator.html

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - If the block satisfies the conditions indicated above this is *True*, else it is *False*.

- **learn(train_data=None, env=None)** - This method first calls the *base* method **learn()** implemented in the Class *Block*. Then it checks that:

  - The current block was fully instantiated: *ModelGeneration* blocks undergo two-stage initialisation and therefore the method **learn()** of a *ModelGeneration* block can be called only once the method **full_block_instantiation** has been called.

    Why is this needed? Because in a pipeline the environment may change and thus also the observation and action space of a RL environment may change, and therefore we don't know the specifics of the environment with which we are working with at the start of the pipeline but only once we reach the *ModelGeneration* block.

    Furthermore the specifics of the environment with which we are working with are needed in order to instantiate MushroomRL algorithms objects.
  - If the **pipeline_type** is *'offline'* the **train_data** must not be *None*.
  - If the **pipeline_type** is *'online'* the **env** must not be *None*.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*BlockOutput*) -This method returns an empty object of Class *BlockOutput* if the call to the *base* method **learn()** implemented in the Class *Block* was not successful.

    Otherwise it returns the **train_data** and the **env**: these are going to be passed onto a specific *ModelGeneration* block.

- **_walk_dict_to_flatten_it(structured_dict, dict_to_fill)** - This method is a recursive method that visits every sub-dictionary in the main dictionary with the aim to create a single flat dictionary from the original one.

  Parameters:

  - **structured_dict** (*dict*) - This is the original starting dictionary: it can contain several sub-dictionaries.
  - **dict_to_fill** (*dict*) - This is a variable that is passed to every recursive call of this method. At the first call to this method it is an empty dictionary.

  Returns:

  - **dict_to_fill** (*dict*) - This is the filled flat dictionary.

- **get_params()** - This method first extracts the **algo_params**: this can be a structured dictionary (i.e: the main dictionary can contain sub-dictionaries), therefore first it is rendered flat. This is achieved by calling the method **_walk_dict_to_flatten_it()**.

  Then a deep copy of this flat dictionary is returned.

  Returns:

  - **flat_dict** (*dict*) - This is a flat dictionary containing a deep copy of the elements present in **algo_params**.

---

14 model_generation_online.py

In the module *model_generation_online.py* the Classes *ModelGenerationMushroomOnline*, *ModelGenerationMushroomOnlineDQN*, *ModelGenerationMushroomOnlineAC*, *ModelGenerationMushroomOnlinePPO*, *ModelGenerationMushroomOnlineSAC*, *ModelGenerationMushroomOnlineDDPG* and *ModelGenerationMushroomOnlineGPOMDP* are implemented.

14.1 ModelGenerationMushroomOnline

This Class is used to contain all the common methods for the online model generation algorithms that are implemented in MushroomRL.

The Class *ModelGenerationMushroomOnline* inherits from the Class *ModelGeneration*. This is an *Abstract Class*.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', deterministic_output_policy=True)

Methods:

- **learn(train_data=None, env=None)** - This method first calls the *base* method **learn()** implemented in the Class *ModelGeneration*. Then:
  - If the call to the *base* method **learn()** implemented in the Class *ModelGeneration* was not successful we return an empty object of Class *BlockOutput*.
  - If the call to the *base* method **learn()** implemented in the Class *ModelGeneration* was successful:
    - If *PyTorch* is being used the number of threads of *PyTorch* is set.

      To determine whether *PyTorch* is being used or not I simply check if the current block has the method **_default_network()**: this is not problematic indeed in MushroomRL you can only use *PyTorch* and not other deep learning frameworks.
    - The **core** is created by calling the method **_create_core()**.
    - The **dict_of_evals** is created and initialised as empty dictionary: this will contain all the evaluation of the RL algorithm performed throughout training.
    - if the parameter **deterministic_output_policy** is equal to *True* then we call the method **make_policy_determi** onto the output block. If this calls happens it will be possible to use the batch evaluation in the metric *DiscountedReward*.
    - The RL algorithm is evaluated on the environment according to the provided **eval_metric**.
    - If the RL algorithm has a replay buffer it is filled with random dataset and the **dict_of_evals** is updated by calling the method **update_dict_of_evals()**.
    - Now we proceed as it is usually done and we perform **n_epochs** in each of which we alternate between:
      - Learning the RL algorithm, by calling the method **learn()** of the member **core**
      - if the parameter **deterministic_output_policy** is equal to *True* then we call the method **make_policy_dete** onto the output block. If this calls happens it will be possible to use the batch evaluation in the metric *DiscountedReward*.
      - Evaluating the RL algorithm according to the provided **eval_metric**.

        Here the **dict_of_evals** is updated by calling the method **update_dict_of_evals()**.

Parameters:

- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- (*BlockOutput*) -This method returns an empty object of Class *BlockOutput* if the call to the *base* method **learn()** implemented in the Class *ModelGeneration* was not successful.

  Otherwise the object of Class *BlockOutput* contains the **policy** which is constructed by calling the method **construct_policy()**.

  If **deterministic_output_policy** is equal to *True* it will be possible to use the batch evaluation in the metric *DiscountedReward*, otherwise the *DiscountedReward* Class can only use non-batch evaluation.

  To see more about what this means see the Class *DiscountedReward*.

- **plot_dict_of_evals()** - This method generates a plot representing how the average discounted reward changes as the agent interacts with more environment steps. It uses the **dict_of_evals** to make the plot. The standard deviation and the mean across the episodes are plotted.

- **update_dict_of_evals(current_epoch, single_episodes_eval, env)** - This method update the **dict_of_evals** which is a dictionary containing as key the current step and as value the average discounted reward of the agent over a number of episodes equal to that specified in the **eval_metric**.

  This is used to keep track of the evolution of the agent evaluation as learning progresses. This method does not use the trajectories that were extracted by the algorithm as part of the learning procedure, but interacts with the environment.

  This method adds a new entry to the dictionary with key given by the number of interactions with the environment happened so far and value equal to the list **single_episodes_eval**.

  The user should not call this method as it is called by the method **learn()** of online model generation blocks.

  Parameters:

  - **current_epoch** (*int*) - This is a non-negative integer and it represents the current epoch: this is used to make the plot, indeed for each epoch we plots the mean evaluation and the standard deviation of the evaluation.
  - **single_episodes_eval** (*list*) - This is a list of floats containing the evaluation of the agent over the single episodes, for as many episodes as specified by the **eval_metric**.
  - **env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

- **_create_core(env)** - This method create an object of Class *mushroom_rl.core.Core* and it assigns it to the **core** member of the **online** *ModelGeneration* object. This is needed in order to run online RL algorithms implemented in MushroomRL.

  Parameters:

  - **env** (*BaseEnvironment*) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

- **analyse()** - This method is not yet implemented. It should analyse the learnt **online** *ModelGeneration* block, namely it should evaluate it according to the provided **eval_metric**.

- **save()** - This method saves the block to a *pickle* file. This method deep copies the current block and then it removes from the deep copy the members **core** and **algo_object**, then it calls the method **save()** of the deep copied object.

This is done because many online *ModelGeneration* blocks have a replay buffer which may contain a lot of data thus causing the *pickle* object obtained from calling the method **save()** be very heavy. This might result problematic when saving several thousand objects in a *Tuner*.

It is the deep copy of the block that is modified and then saved, indeed we cannot modify directly the object itself because the member **algo_object** is not restored in the call of the method **learn()**: the member **algo_object** is restored in the call of the method **set_params()** hence we would not be able to call twice in a row the method **learn()** if we were to modify directly the object itself, as we would not have the right value for **algo_object**.

14.2 ModelGenerationMushroomOnlineDQN

This Class implements a specific online model generation algorithm: DQN. This Class wraps the DQN method implemented in MushroomRL.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/algorithms/value/dqn/dqn.py

The Class *ModelGenerationMushroomOnlineDQN* inherits from the Class *ModelGenerationMushroomOnline*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*.
- **works_on_offline_rl** is equal to *False*.
- **works_on_box_action_space** is equal to *False*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'online'*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, regressor_type='q_regressor', seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', deterministic_output_policy=True)

Parameters:

- **regressor_type** (*str*, 'q_regressor') - This is a string and it used to pick between the three possible regressors that can be used in MushroomRL. For more information about the different regressor see the link in the explanation of method **pre_learn_check()** of the *ModelGeneration* Class.
- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:

  - 'epsilon': mushroom_rl.utils.parameters.LinearParameter(value=1, threshold_value=0.01, n=1000000)
  - 'policy': mushroom_rl.policy.EpsGreedy(epsilon='epsilon')
  - 'approximator': mushroom_rl.approximators.parametric.TorchApproximator
  - 'network': one hidden layer made up of 16 neurons and ReLU as activation function.
  - 'optimizer': torch.optim.Adam
  - 'lr': 0.0001
  - 'critic_loss': torch.nn.functional.smooth_l1_loss
  - 'batch_size': 32
  - 'target_update_frequency': 250
  - 'replay_memory': mushroom_rl.utils.replay_memory.ReplayMemory
  - 'initial_replay_size': 50000
  - 'max_replay_size': 1000000

- 'clip_reward': False
- 'n_epochs': 10
- 'n_steps': None
- 'n_steps_per_fit': None
- 'n_episodes': 500
- 'n_episodes_per_fit': 50

- **deterministic_output_policy** (*bool*, True) - This is a boolean and if *True* the method **make_policy_deterministic()**, of the Class *BlockOutput*, will be called before the evaluation step through the method **learn()** of this block. This turns the policy contained in an object of Class *BlockOutput* into a deterministic policy.

  If this happens then the mode **batch** evaluation can be used in the Class *DiscountedReward*: to see more about what this means see the Class *DiscountedReward*.

Non-Parameters Members:

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.
- **algo_object** (None) - This is the object containing the actual RL online algorithm. In this case it is an object of Class *mushroom_rl.algorithms.value.dqn.DQN*.
- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **model** - This contains the *Class* of the actual RL online algorithm. In this case it contains the Class *mushroom_rl.algorithms.value.dqn.DQN*.
- **core** (*mushroom_rl.core.Core*, None) - This contains the **Core** object of MushroomRL that is needed in order to run an online RL algorithm.
- **dict_of_evals** (*dict*) - This is a dictionary containing as key the current step and as value the average reward of the agent over the last 10 episodes ending at said step. This is used to keep track of the evolution of the agent evaluation as learning progresses.

Methods:

- **_default_network()** - This method creates a Class inheriting from the Class *torch.nn.Module* that has one hidden layer made up of 16 neurons with activation function ReLU.

  This network will be used in DQN.

- **full_block_instantiation(info_MDP)** - This method sets **info_MDP** equal to **info_MDP**. Then if **algo_params** is *None* the default dictionary of parameters to be used in DQN is created (as specified above in the **Parameters** section) and this default dictionary is assigned to **algo_params**.

  Now the method **set_params()** is called by passing the member **algo_params**. If the parameters were set successfully this method returns *True*, else it returns *False*.

  Parameters:

- **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class *mushroom_rl.environment.MDPInfo* containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Core*.

Returns:

- (*bool*) - This is *True* if the block was successfully fully instantiated, else it is *False*.

- **set_params(new_params)** - This method changes the value of **algo_params** and of **algo_params_upon _instantiation** to that of **new_params**. Moreover:

  - The **policy**, **input_shape**, **output_shape** and **n_actions** variables are created: these are needed for instantiating a **core** object from MushroomRL.

    The **policy** is an epsilon greedy policy, while the other variables are created according to the choice of **regressor_type**. To see how read through the web page linked in the explanation of the method **pre_learn_check()** of the Class *ModelGeneration*.
  - A structured dictionary is created: this is needed in order to properly instantiate a MushroomRL **agent** object. Moreover in order to instantiate such an object we also need to extract the **current_actual_value** from the **new_params** dictionary, which is made of *HyperParameter* objects.
  - An object of Class *mushroom_rl.algorithms.value.dqn.DQN* is created passing to it the structured dictionary created in the previous step. This is saved in the member **algo_object**.

Parameters:

- **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.

Returns:

- (*bool*) - This is *True* if **new_params** was set successfully, else it is *False*.

14.3 ModelGenerationMushroomOnlineAC

This Class is used as base Class for actor-critic methods implemented in MushroomRL. Specifically is used to contain some common methods that would have the same implementation across different actor critic methods.

The Class *ModelGenerationMushroomOnlineAC* inherits from the Class *ModelGenerationMushroomOnline*. This is an *Abstract Class*.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', deterministic_output_policy=True)

Methods:

- **set_params(new_params)** - This method changes the value of **algo_params** and of **algo_params_upon_instantiation** to that of **new_params**. Moreover:
  - The **input_shape**, **output_shape** and **n_actions** variables are created: these are needed for instantiating a **core** object from MushroomRL.

    These variables are created according to the choice of **regressor_type**. To see how read through the web page linked in the explanation of the method **pre_learn_check()** of the Class *ModelGeneration*.
  - The method **model_specific_set_params()** is called: this method is implemented in Classes that inherit from this Class. This method:
    * Returns a structured dictionary: this is needed in order to properly instantiate a MushroomRL **agent** object. Moreover in order to instantiate such an object we also need to extract the **current_actual_value** from the **new_params** dictionary, which is made of *HyperParameter* objects.
    * An object of Class **model** is created passing to it the structured dictionary created in the previous step. This is saved in the member **algo_object**.

      The Class **model** is specified in Class that inherit from this Class.

  Parameters:

  - **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.

  Returns:

  - (*bool*) - This is *True* if **new_params** was set successfully, else it is *False*.

14.4 ModelGenerationMushroomOnlinePPO

This Class implements a specific online model generation algorithm: PPO. This Class wraps the PPO method implemented in MushroomRL.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/algorithms/actor_critic/deep_actor_critic/ppo.py

The Class *ModelGenerationMushroomOnlinePPO* inherits from the Class *ModelGenerationMushroomOnlineAC*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*.
- **works_on_offline_rl** is equal to *False*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'online'*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, regressor_type='generic_regressor', seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', deterministic_output_policy=True)

Parameters:

- **regressor_type** (*str*, 'generic_regressor') - This is a string and it used to pick between the three possible regressors that can be used in MushroomRL. For more information about the different regressor see the link in the explanation of method **pre_learn_check()** of the *ModelGeneration* Class.
- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:

  - 'policy':
    - For *Box* action spaces: mushroom_rl.policy.GaussianTorchPolicy(std_0=1)
    - For *Discrete* action spaces: mushroom_rl.policy.BoltzmannTorchPolicy(beta=0.001)
  - 'network': one hidden layer made up of 16 neurons and ReLU as activation function.
  - 'actor_class': torch.optim.Adam
  - 'actor_lr': 0.0003
  - 'critic_class': torch.optim.Adam
  - 'critic_lr': 0.0003
  - 'loss': torch.nn.functional.mse_loss
  - 'n_epochs_policy': 10
  - 'batch_size': 64

- 'eps_ppo': 0.2
- 'lam': 0.95
- 'ent_coeff': 0
- 'n_epochs': 10
- 'n_steps': None
- 'n_steps_per_fit': None
- 'n_episodes': 500
- 'n_episodes_per_fit': 50

- **deterministic_output_policy** (*bool*, True) - This is a boolean and if *True* the method **make_policy_deterministic()**, of the Class *BlockOutput*, will be called before the evaluation step through the method **learn()** of this block. This turns the policy contained in an object of Class *BlockOutput* into a deterministic policy.

  If this happens then the mode **batch** evaluation can be used in the Class *DiscountedReward*: to see more about what this means see the Class *DiscountedReward*.

Non-Parameters Members:

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.
- **algo_object** (None) - This is the object containing the actual RL online algorithm. In this case it is an object of Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.PPO*.
- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **model** - This contains the *Class* of the actual RL online algorithm. In this case it contains the Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.PPO*.
- **core** (*mushroom_rl.core.Core*, None) - This contains the **Core** object of MushroomRL that is needed in order to run an online RL algorithm.
- **dict_of_evals** (*dict*) - This is a dictionary containing as key the current step and as value the average reward of the agent over the last 10 episodes ending at said step. This is used to keep track of the evolution of the agent evaluation as learning progresses.

Methods:

- **_default_network()** - This method creates a Class inheriting from the Class *torch.nn.Module* that has one hidden layer made up of 16 neurons with activation function ReLU.

  This network will be used in PPO.

- **full_block_instantiation(info_MDP)** - This method sets **info_MDP** equal to **info_MDP**. Then if **algo_params** is *None* the default dictionary of parameters to be used in PPO is created (as specified above in the **Parameters** section) and this default dictionary is assigned to **algo_params**.

  Now the method **set_params()** is called by passing the member **algo_params**. If the parameters were set successfully this method returns *True*, else it returns *False*.

  Parameters:

- **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class *mushroom_rl.environment.MDPInfo* containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Core*.

Returns:

- (*bool*) - This is *True* if the block was successfully fully instantiated, else it is *False*.

- **model_specific_set_params(new_params, mdp_info, input_shape, output_shape, n_actions)** - This method is called by the method **set_params()** in the common parent Class *ModelGenerationMushroomOnlineAC*.

This method:

- Selects the type of **policy** based on the action space, as described above in the **Parameters** section under the voice **algo_params**.
- Starting from the flat dictionary **new_params** it constructs the proper structured dictionary that MushroomRL expects.

  This structured dictionary is made up of *HyperParameter* objects: in order to instantiate a MushroomRL **agent** object we need to extract the **current_actual_value** from the newly created structured dictionary.
- An object of Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.PPO* is created passing to it the structured dictionary created in the previous step. This is saved in the member **algo_object**.

This method should not be called by the user: it is called in the parent Class *ModelGenerationMushroomOnlineAC*.

Parameters:

- **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.
- **mdp_info** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class *mushroom_rl.environment.MDPInfo* containing the observation space, the action space, gamma and the horizon of the environment. This is needed for selecting the **policy** to use: from **mdp_info** the action space is extracted so that we know if it is *Box* or *Discrete*.
- **input_shape** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

  This is the input shape of the environment, namely the shape of the observation space. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.
- **output_shape** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

  This is the output shape of the environment, namely the shape of the action space. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.
- **n_actions** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

  In case the environment has *Discrete* action space this is used to represent the number of actions. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.

Returns:

- **tmp_structured_algo_params** (*dict*) - This is a structured dictionary containing *HyperParameter* objects: it compromises all the parameters needed to instantiate a MushroomRL **agent** object.
- **dict_to_add** (*dict*) - This is a dictionary is a flat dictionary containing more parameters that are not needed for the MushroomRL **agent** object but that are needed for the overall *ModelGeneration* block. Examples are:

* The number of episodes and the number of episodes per fit.
* The number of epochs: for how many times do we need to alternate between learning the agent and evaluating it?

14.5 ModelGenerationMushroomOnlineSAC

This Class implements a specific online model generation algorithm: SAC. This Class wraps the SAC method implemented in MushroomRL.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/algorithms/actor_critic/deep_actor_critic/sac.py

The Class *ModelGenerationMushroomOnlineSAC* inherits from the Class *ModelGenerationMushroomOnlineAC*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*.
- **works_on_offline_rl** is equal to *False*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *False*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'online'*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, regressor_type='generic_regressor', seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', deterministic_output_policy=True)

Parameters:

- **regressor_type** (*str*, 'generic_regressor') - This is a string and it used to pick between the three possible regressors that can be used in MushroomRL. For more information about the different regressor see the link in the explanation of method **pre_learn_check()** of the *ModelGeneration* Class.
- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:

  - 'actor_network_mu': one hidden layer made up of 16 neurons and ReLU as activation function.
  - 'actor_network_sigma': one hidden layer made up of 16 neurons and ReLU as activation function.
  - 'critic_network': one hidden layer made up of 16 neurons and ReLU as activation function.
  - 'actor_class': torch.optim.Adam
  - 'actor_lr': 0.0003
  - 'critic_class': torch.optim.Adam
  - 'critic_lr': 0.0003
  - 'loss': torch.nn.functional.mse_loss
  - 'batch_size': 256
  - 'initial_replay_size': 50000
  - 'max_replay_size': 1000000

- 'warmup_transitions': 100
- 'tau': 0.005
- 'lr_alpha': 0.0003
- 'log_std_min': -20
- 'log_std_max': 2
- 'target_entropy': None
- 'n_epochs': 10
- 'n_steps': None
- 'n_steps_per_fit': None
- 'n_episodes': 500
- 'n_episodes_per_fit': 50

- **deterministic_output_policy** (*bool*, True) - This is a boolean and if *True* the method **make_policy_deterministic()**, of the Class *BlockOutput*, will be called before the evaluation step through the method **learn()** of this block. This turns the policy contained in an object of Class *BlockOutput* into a deterministic policy.

  If this happens then the mode **batch** evaluation can be used in the Class *DiscountedReward*: to see more about what this means see the Class *DiscountedReward*.

Non-Parameters Members:

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.
- **algo_object** (None) - This is the object containing the actual RL online algorithm. In this case it is an object of Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.SAC*.
- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **model** - This contains the *Class* of the actual RL online algorithm. In this case it contains the Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.SAC*.
- **core** (*mushroom_rl.core.Core*, None) - This contains the **Core** object of MushroomRL that is needed in order to run an online RL algorithm.
- **dict_of_evals** (*dict*) - This is a dictionary containing as key the current step and as value the average reward of the agent over the last 10 episodes ending at said step. This is used to keep track of the evolution of the agent evaluation as learning progresses.

Methods:

- **_default_network()** - This method creates two Classes, both inheriting from the Class *torch.nn.Module*: one is the critic network, the other is the actor network.

  Both networks have one hidden layer made up of 16 neurons with activation function ReLU.

  These networks will be used in SAC.

- **full_block_instantiation(info_MDP)** - This method sets **info_MDP** equal to **info_MDP**. Then if **algo_params** is *None* the default dictionary of parameters to be used in SAC is created (as specified above in the **Parameters** section) and this default dictionary is assigned to **algo_params**.

  Now the method **set_params()** is called by passing the member **algo_params**. If the parameters were set successfully this method returns *True*, else it returns *False*.

  Parameters:

  - **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class *mushroom_rl.environment.MDPInfo* containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Core*.

  Returns:

  - (*bool*) - This is *True* if the block was successfully fully instantiated, else it is *False*.

- **model_specific_set_params(new_params, mdp_info, input_shape, output_shape, n_actions)** - This method is called by the method **set_params()** in the common parent Class *ModelGenerationMushroomOnlineAC*.

  This method:

  - Starting from the flat dictionary **new_params** it constructs the proper structured dictionary that MushroomRL expects.

    This structured dictionary is made up of *HyperParameter* objects: in order to instantiate a MushroomRL **agent** object we need to extract the **current_actual_value** from the newly created structured dictionary.
  - An object of Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.SAC* is created passing to it the structured dictionary created in the previous step. This is saved in the member **algo_object**.

  This method should not be called by the user: it is called in the parent Class *ModelGenerationMushroomOnlineAC*.

  Parameters:

  - **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.
  - **mdp_info** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class *mushroom_rl.environment.MDPInfo* containing the observation space, the action space, gamma and the horizon of the environment. This is needed for selecting the **policy** to use: from **mdp_info** the action space is extracted so that we know if it is *Box* or *Discrete*.
  - **input_shape** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    This is the input shape of the environment, namely the shape of the observation space. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.
  - **output_shape** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    This is the output shape of the environment, namely the shape of the action space. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.
  - **n_actions** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    In case the environment has *Discrete* action space this is used to represent the number of actions. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.

Returns:

- **tmp_structured_algo_params** (*dict*) - This is a structured dictionary containing *HyperParameter* objects: it compromises all the parameters needed to instantiate a MushroomRL **agent** object.
- **dict_to_add** (*dict*) - This is a dictionary is a flat dictionary containing more parameters that are not needed for the MushroomRL **agent** object but that are needed for the overall *ModelGeneration* block. Examples are:
  - The number of episodes and the number of episodes per fit.
  - The number of epochs: for how many times do we need to alternate between learning the agent and evaluating it?

14.6 ModelGenerationMushroomOnlineDDPG

This Class implements a specific online model generation algorithm: DDPG. This Class wraps the DDPG method implemented in MushroomRL.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/algorithms/actor_critic/deep_actor_critic/ddpg.py

The Class *ModelGenerationMushroomOnlineDDPG* inherits from the Class *ModelGenerationMushroomOnlineAC*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*.
- **works_on_offline_rl** is equal to *False*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *False*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'online'*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, regressor_type='generic_regressor', seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', deterministic_output_policy=True)

Parameters:

- **regressor_type** (*str*, 'generic_regressor') - This is a string and it used to pick between the three possible regressors that can be used in MushroomRL. For more information about the different regressor see the link in the explanation of method **pre_learn_check()** of the *ModelGeneration* Class.
- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:

  - 'policy': mushroom_rl.policy.noise_policy.OrnsteinUhlenbeckPolicy(sigma, theta, dt) where:
    * sigma = 0.2*np.ones(1)
    * theta = 0.15
    * dt = 1e-2
  - 'actor_network': one hidden layer made up of 16 neurons and ReLU as activation function.
  - 'critic_network': one hidden layer made up of 16 neurons and ReLU as activation function.
  - 'actor_class': torch.optim.Adam
  - 'actor_lr': 0.001
  - 'critic_class': torch.optim.Adam
  - 'critic_lr': 0.001
  - 'loss': torch.nn.functional.mse_loss

    - 'batch_size': 100
    - 'initial_replay_size': 50000
    - 'max_replay_size': 1000000
    - 'tau': 0.005
    - 'policy_delay': 1
    - 'n_epochs': 10
    - 'n_steps': None
    - 'n_steps_per_fit': None
    - 'n_episodes': 500
    - 'n_episodes_per_fit': 50

- **deterministic_output_policy** (*bool*, True) - This is a boolean and if *True* the method **make_policy_deterministic()**, of the Class *BlockOutput*, will be called before the evaluation step through the method **learn()** of this block. This turns the policy contained in an object of Class *BlockOutput* into a deterministic policy.

  If this happens then the mode **batch** evaluation can be used in the Class *DiscountedReward*: to see more about what this means see the Class *DiscountedReward*.

Non-Parameters Members:

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.
- **algo_object** (None) - This is the object containing the actual RL online algorithm. In this case it is an object of Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.DDPG*.
- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **model** - This contains the *Class* of the actual RL online algorithm. In this case it contains the Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.DDPG*.
- **core** (*mushroom_rl.core.Core*, None) - This contains the **Core** object of MushroomRL that is needed in order to run an online RL algorithm.
- **dict_of_evals** (*dict*) - This is a dictionary containing as key the current step and as value the average reward of the agent over the last 10 episodes ending at said step. This is used to keep track of the evolution of the agent evaluation as learning progresses.

Methods:

- **_default_network()** - This method creates two Classes, both inheriting from the Class *torch.nn.Module*: one is the critic network, the other is the actor network.

  Both networks have one hidden layer made up of 16 neurons with activation function ReLU.

  These networks will be used in DDPG.

- **full_block_instantiation(info_MDP)** - This method sets **info_MDP** equal to **info_MDP**. Then if **algo_params** is *None* the default dictionary of parameters to be used in DDPG is created (as specified above in the **Parameters** section) and this default dictionary is assigned to **algo_params**.

  Now the method **set_params()** is called by passing the member **algo_params**. If the parameters were set successfully this method returns *True*, else it returns *False*.

  Parameters:

  - **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class *mushroom_rl.environment.MDPInfo* containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Core*.

  Returns:

  - (*bool*) - This is *True* if the block was successfully fully instantiated, else it is *False*.

- **model_specific_set_params(new_params, mdp_info, input_shape, output_shape, n_actions)** - This method is called by the method **set_params()** in the common parent Class *ModelGenerationMushroomOnlineAC*.

  This method:

  - Starting from the flat dictionary **new_params** it constructs the proper structured dictionary that MushroomRL expects.

    This structured dictionary is made up of *HyperParameter* objects: in order to instantiate a MushroomRL **agent** object we need to extract the **current_actual_value** from the newly created structured dictionary.
  - An object of Class *mushroom_rl.algorithms.actor_critic.deep_actor_critic.DDPG* is created passing to it the structured dictionary created in the previous step. This is saved in the member **algo_object**.

  This method should not be called by the user: it is called in the parent Class *ModelGenerationMushroomOnlineAC*.

  Parameters:

  - **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.
  - **mdp_info** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class *mushroom_rl.environment.MDPInfo* containing the observation space, the action space, gamma and the horizon of the environment. This is needed for selecting the **policy** to use: from **mdp_info** the action space is extracted so that we know if it is *Box* or *Discrete*.
  - **input_shape** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    This is the input shape of the environment, namely the shape of the observation space. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.
  - **output_shape** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    This is the output shape of the environment, namely the shape of the action space. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.
  - **n_actions** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    In case the environment has *Discrete* action space this is used to represent the number of actions. This parameter is set in the parent Class *ModelGenerationMushroomOnlineAC* based on the value of **regressor_type**.

Returns:

- **tmp_structured_algo_params** (*dict*) - This is a structured dictionary containing *HyperParameter* objects: it compromises all the parameters needed to instantiate a MushroomRL **agent** object.
- **dict_to_add** (*dict*) - This is a dictionary is a flat dictionary containing more parameters that are not needed for the MushroomRL **agent** object but that are needed for the overall *ModelGeneration* block. Examples are:
  * The number of episodes and the number of episodes per fit.
  * The number of epochs: for how many times do we need to alternate between learning the agent and evaluating it?

14.7 ModelGenerationMushroomOnlineGPOMDP

This Class implements a specific online model generation algorithm: GPOMDP. This Class wraps the GPOMDP method implemented in MushroomRL.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/algorithms/policy_search/policy_gradient/gpomdp.py

The Class *ModelGenerationMushroomOnlineGPOMDP* inherits from the Class *ModelGenerationMushroomOnline*.

Initialiser:
__init__(eval_metric, obj_name, regressor_type='generic_regressor', seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', deterministic_output_policy=True)

Parameters:

- **regressor_type** (*str*, 'generic_regressor') - This is a string and it used to pick between the three possible regressors that can be used in MushroomRL. For more information about the different regressor see the link in the explanation of method **pre_learn_check()** of the *ModelGeneration* Class.
- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:

  - 'policy': mushroom_rl.policy.gaussian_policy.StateStdGaussianPolicy
  - 'approximator': mushroom_rl.approximators.parametric.linear.LinearApproximator
  - 'optimizer': mushroom_rl.utils.optimizers.AdaptiveOptimizer
  - 'eps': 1e-2
  - 'n_epochs': 10
  - 'n_steps': None
  - 'n_steps_per_fit': None
  - 'n_episodes': 500
  - 'n_episodes_per_fit': 50
- **deterministic_output_policy** (*bool*, True) - This is a boolean and if *True* the method **make_policy_deterministic()**, of the Class *BlockOutput*, will be called before the evaluation step through the method **learn()** of this block. This turns the policy contained in an object of Class *BlockOutput* into a deterministic policy.

  If this happens then the mode **batch** evaluation can be used in the Class *DiscountedReward*: to see more about what this means see the Class *DiscountedReward*.

Non-Parameters Members:

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.
- **algo_object** (None) - This is the object containing the actual RL online algorithm. In this case it is an object of Class *mushroom_rl.algorithms.policy_search.policy_gradient.GPOMDP*.

- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **model** - This contains the *Class* of the actual RL online algorithm. In this case it contains the Class *mushroom_rl.algorithms.policy_search.policy_gradient.GPOMDP*.
- **core** (*mushroom_rl.core.Core*, None) - This contains the **Core** object of MushroomRL that is needed in order to run an online RL algorithm.
- **dict_of_evals** (*dict*) - This is a dictionary containing as key the current step and as value the average reward of the agent over the last 10 episodes ending at said step. This is used to keep track of the evolution of the agent evaluation as learning progresses.

Methods:

- **full_block_instantiation(info_MDP)** - This method sets **info_MDP** equal to **info_MDP**. Then if **algo_params** is *None* the default dictionary of parameters to be used in GPOMDP is created (as specified above in the **Parameters** section) and this default dictionary is assigned to **algo_params**.

  Now the method **set_params()** is called by passing the member **algo_params**. If the parameters were set successfully this method returns *True*, else it returns *False*.

  Parameters:

  - **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class *mushroom_rl.environment.MDPInfo* containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Core*.

  Returns:

  - (*bool*) - This is *True* if the block was successfully fully instantiated, else it is *False*.

- **_create_policy(input_shape, n_actions, output_shape)** - This method create the default policy: a policy of Class *mushroom_rl.policy.gaussian_policy.StateStdGaussianPolicy*.

  You can create a Class inheriting from this Class and override this method to create a custom policy. The policy cannot be specified via a parameter because it can only be constructed at run time: we don't have the input parameters of this method when an object of this Class is initialised.

  Parameters:

  - **input_shape** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    This is the input shape of the environment, namely the shape of the observation space.
  - **output_shape** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    This is the output shape of the environment, namely the shape of the action space.
  - **n_actions** (*HyperParameter*) - This is an object of a Class inheriting from the Class *HyperParameter*.

    In case the environment has *Discrete* action space this is used to represent the number of actions.

  Returns:

  - **policy** (*Categorical*) - This is an object of Class *Categorical* and it contains the policy: an object of a Class inheriting from the Class *mushroom_rl.policy.Policy*.

- **set_params(new_params)** - This method changes the value of **algo_params** and of **algo_params_upon _instantiation** to that of **new_params**. Moreover:
  - The **policy**, **input_shape**, **output_shape** and **n_actions** variables are created: these are needed for instantiating a **core** object from MushroomRL.

    The **policy** is create by calling the method **_create_policy()**, while the other variables are created according to the choice of **regressor_type**. To see how read through the web page linked in the explanation of the method **pre_learn_check()** of the Class *ModelGeneration*.
  - A structured dictionary is created: this is needed in order to properly instantiate a MushroomRL **agent** object. Moreover in order to instantiate such an object we also need to extract the **current_actual_value** from the **new_params** dictionary, which is made of *HyperParameter* objects.
  - An object of Class *mushroom_rl.algorithms.policy_search.policy_gradient.GPOMDP* is created passing to it the structured dictionary created in the previous step. This is saved in the member **algo_object**.

  Parameters:

  - **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.

  Returns:

  - (*bool*) - This is *True* if **new_params** was set successfully, else it is *False*.

---

15 model_generation_offline.py

In the module *model_generation_offline.py* the Classes *ModelGenerationMushroomOffline*, *ModelGenerationMushroomOfflineFQI*, *ModelGenerationMushroomOfflineDoubleFQI* and *ModelGenerationMushroomOfflineLSPI* are implemented.

15.1 ModelGenerationMushroomOffline

This Class is used to contain all the common methods for the offline model generation algorithms that are implemented in MushroomRL.

The Class *ModelGenerationMushroomOffline* inherits from the Class *ModelGeneration* .This is an *Abstract Class*.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **learn(train_data=None, env=None)** - This method first calls the *base* method **learn()** implemented in the Class *ModelGeneration*. Then:
  - If the call to the *base* method **learn()** implemented in the Class *ModelGeneration* was not successful we return an empty object of Class *BlockOutput*.
  - If the call to the *base* method **learn()** implemented in the Class *ModelGeneration* was successful we call the method **fit()** of the **algo_object** on the member **dataset** of the **train_data**.

  Parameters:
  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:
  - This method returns an empty object of Class *BlockOutput* if the call to the *base* method **learn()** implemented in the Class *ModelGeneration* was not successful.

    Otherwise the object of Class *BlockOutput* contains the **policy** which is constructed by calling the method **construct_policy()**.

    Here the **approximator** parameter is passed to the method **construct_policy()** therefore the *DiscountedReward* Class can use both batch and non-batch evaluation on blocks of a Class inheriting from this Class.

    To see more about what this means see the Class *DiscountedReward*.

- **set_params(new_params)** - This method changes the value of **algo_params** and of **algo_params_upon_instantiation** to that of **new_params**. Moreover:
  - The **policy**, **input_shape**, **output_shape** and **n_actions** variables are created: these are needed for instantiating the actual offline RL algorithm from MushroomRL.

    The **policy** is an epsilon greedy policy, while the other variables are created according to the choice of **regressor_type**. To see how read the web page linked in the explanation of the method **pre_learn_check()** of the Class *ModelGeneration*.

- A structured dictionary is created: this is needed in order to properly instantiate a MushroomRL **agent** object. Moreover in order to instantiate such an object we also need to extract the **current_actual_value** from the **new_params** dictionary, which is made of *HyperParameter* objects.
- An object of Class *mushroom_rl.core.Agent* (based on the value of the member **model**) is created passing to it the structured dictionary created in the previous step. This is saved in the member **algo_object**.

Parameters:

- **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.

Returns:

- (*bool*) - This is *True* if **new_params** was set successfully, else it is *False*.

- **analyse()** - This method is not yet implemented. It should analyse the learnt **offline** *ModelGeneration* block, namely it should evaluate it according to the provided **eval_metric**.

- **save()** - This method saves the block to a *pickle* file. This method deep copies the current block and then it removes from the deep copy the member **algo_object**, then it calls the method **save()** of the deep copied object.

  This is done because the **algo_object** may contain a lot of data thus causing the *pickle* object obtained from calling the method **save()** be very heavy. This might result problematic when saving several thousand objects in a *Tuner*.

  It is the deep copy of the block that is modified and then saved, indeed we cannot modify directly the object itself because the member **algo_object** is not restored in the call of the method **learn()**: the member **algo_object** is restored in the call of the method **set_params()** hence we would not be able to call twice in a row the method **learn()** if we were to modify directly the object itself, as we would not have the right value for **algo_object**.

15.2 ModelGenerationMushroomOfflineFQI

This Class implements a specific offline model generation algorithm: FQI. This class wraps the Fitted Q-Iteration method implemented in MushroomRL.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/algorithms/value/batch_td/fqi.py

The Class *ModelGenerationMushroomOfflineFQI* inherits from the Class *ModelGenerationMushroomOffline*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *False*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'offline'*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, regressor_type='action_regressor', n_train_samples=None, seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:

  - 'policy': mushroom_rl.policy.EpsGreedy(value=0)
  - 'approximator': XGBRegressor
  - 'n_iterations': 10
  - 'n_estimators': 300
  - 'subsample': 0.8
  - 'colsample_bytree': 0.3
  - 'colsample_bylevel': 0.7
  - 'learning_rate': 0.08
  - 'verbosity': 0
  - 'random_state': 3
  - 'n_jobs': 1

- **regressor_type** (*str*, 'action_regressor') - This is a string and it can either be: *'action_regressor'*, *'q_regressor'* or *'generic_regressor'*. This is used to pick one of the 3 possible kind of regressor made available by MushroomRL.

  Note that if you want to use a *'q_regressor'* then the picked regressor must be able to perform multi-target regression, as a single regressor is used for all actions.

- **n_train_samples** (*Integer*, None) - This must be an object of Class Integer (sub-Class of HyperParameter) and it represents a tunable parameter. This has effect only when using the input loader: *LoadDifferentSizeForEachBlock* and *LoadDifferentSizeForEachBlockAndEnv*.

  For this to work the block must have **is_parametrised** equal to *True*, otherwise it does not reach the method **tune()** of the *Tuner*!

  If *None* then an *Integer* object is created with **current_actual_value** equal to 10000 and with **range_of_values** equal to $[100, 1000000]$

Non-Parameters Members:

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.
- **algo_object** (None) - This is the object containing the actual RL offline algorithm. In this case it is an object of Class *mushroom_rl.algorithms.value.batch_td.fqi.FQI*.
- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **model** - This contains the *Class* of the actual RL offline algorithm. In this case it contains the Class *mushroom_rl.algorithms.value.batch_td.fqi.FQI*.

Methods:

- **full_block_instantiation(info_MDP)** - This method sets **info_MDP** equal to **info_MDP**. Then if **algo_params** is *None* the default dictionary of parameters to be used in FQI is created (as specified above in the **Parameters** section) and this default dictionary is assigned to **algo_params**.

  Now the method **set_params()** is called by passing the member **algo_params**. If the parameters were set successfully this method returns *True*, else it returns *False*.

  Parameters:

  – **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Agent*.

  Returns:

  – (*bool*) - This is *True* if the block was successfully fully instantiated, else it is *False*.

15.3 ModelGenerationMushroomOfflineDoubleFQI

This Class implements a specific offline model generation algorithm: DoubleFQI. This class wraps the Double Fitted Q-Iteration method implemented in MushroomRL.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/algorithms/value/batch_td/double_fqi.py

The Class *ModelGenerationMushroomOfflineDoubleFQI* inherits from the Class *ModelGenerationMushroomOffline*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *False*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'offline'*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, regressor_type='action_regressor', n_train_samples=None, seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:

  - 'policy': mushroom_rl.policy.EpsGreedy(value=0)
  - 'approximator': XGBRegressor
  - 'n_iterations': 10
  - 'n_estimators': 300
  - 'subsample': 0.8
  - 'colsample_bytree': 0.3
  - 'colsample_bylevel': 0.7
  - 'learning_rate': 0.08
  - 'verbosity': 0
  - 'random_state': 3
  - 'n_jobs': 1

- **regressor_type** (*str*, 'action_regressor') - This is a string and it can either be: *'action_regressor'*, *'q_regressor'* or *'generic_regressor'*. This is used to pick one of the 3 possible kind of regressor made available by MushroomRL.

  Note that if you want to use a *'q_regressor'* then the picked regressor must be able to perform multi-target regression, as a single regressor is used for all actions.

- **n_train_samples** (*Integer*, None) - This must be an object of Class Integer (sub-Class of HyperParameter) and it represents a tunable parameter. This has effect only when using the input loader: *LoadDifferentSizeForEachBlock* and *LoadDifferentSizeForEachBlockAndEnv*.

  For this to work the block must have **is_parametrised** equal to *True*, otherwise it does not reach the method **tune()** of the *Tuner*!

  If *None* then an *Integer* object is created with **current_actual_value** equal to 10000 and with **range_of_values** equal to $[100, 1000000]$

Non-Parameters Members:

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.
- **algo_object** (None) - This is the object containing the actual RL offline algorithm. In this case it is an object of Class *mushroom_rl.algorithms.value.batch_td.double_fqi.DoubleFQI*.
- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **model** - This contains the *Class* of the actual RL offline algorithm. In this case it contains the Class *mushroom_rl.algorithms.value.batch_td.double_fqi.DoubleFQI*.

Methods:

- **full_block_instantiation(info_MDP)** - This method sets **info_MDP** equal to **info_MDP**. Then if **algo_params** is *None* the default dictionary of parameters to be used in DoubleFQI is created (as specified above in the **Parameters** section) and this default dictionary is assigned to **algo_params**.

  Now the method **set_params()** is called by passing the member **algo_params**. If the parameters were set successfully this method returns *True*, else it returns *False*.

  Parameters:

  – **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Agent*.

  Returns:

  – (*bool*) - This is *True* if the block was successfully fully instantiated, else it is *False*.

15.4 ModelGenerationMushroomOfflineLSPI

This Class implements a specific offline model generation algorithm: LSPI. This class wraps the Least-Squares Policy Iteration method implemented in MushroomRL.
cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/algorithms/value/batch_td/lspi.py

The Class *ModelGenerationMushroomOfflineLSPI* inherits from the Class *ModelGenerationMushroomOffline*.

Note that in this Class:

- **works_on_online_rl** is equal to *False*.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *False*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'offline'*.
- **is_parametrised** is equal to *True*, indeed this block has several parameters that can be tuned.

Initialiser:
__init__(eval_metric, obj_name, regressor_type='action_regressor', n_train_samples=None, seeder=2, algo_params=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **algo_params** (*dict*, None) - This must be a flat dictionary containing all the parameters needed for this block.

  If *None* is provided then the following parameters will be used:

  - 'policy': mushroom_rl.policy.EpsGreedy(value=0)
  - 'epsilon': 1e-2

- **regressor_type** (*str*, 'action_regressor') - This is a string and it can either be: *'action_regressor'*, *'q_regressor'* or *'generic_regressor'*. This is used to pick one of the 3 possible kind of regressor made available by MushroomRL.

  Note that if you want to use a *'q_regressor'* then the picked regressor must be able to perform multi-target regression, as a single regressor is used for all actions.

- **n_train_samples** (*Integer*, None) - This must be an object of Class Integer (sub-Class of HyperParameter) and it represents a tunable parameter. This has effect only when using the input loader: *LoadDifferentSizeForEachBlock* and *LoadDifferentSizeForEachBlockAndEnv*.

  For this to work the block must have **is_parametrised** equal to *True*, otherwise it does not reach the method **tune()** of the *Tuner*!

  If *None* then an *Integer* object is created with **current_actual_value** equal to 10000 and with **range_of_values** equal to $[100, 1000000]$

Non-Parameters Members:

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.
- **algo_object** (None) - This is the object containing the actual RL offline algorithm. In this case it is an object of Class *mushroom_rl.algorithms.value.batch_td.lspi.LSPI*.
- **algo_params_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **algo_params**.
- **model** - This contains the *Class* of the actual RL offline algorithm. In this case it contains the Class *mushroom_rl.algorithms.value.batch_td.lspi.LSPI*.

Methods:

- **full_block_instantiation(info_MDP)** - This method sets **info_MDP** equal to **info_MDP**. Then if **algo_params** is *None* the default dictionary of parameters to be used in LSPI is created (as specified above in the **Parameters** section) and this default dictionary is assigned to **algo_params**.

  Now the method **set_params()** is called by passing the member **algo_params**. If the parameters were set successfully this method returns *True*, else it returns *False*.

  Parameters:

  - **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Agent*.

  Returns:

  - (*bool*) - This is *True* if the block was successfully fully instantiated, else it is *False*.

- **set_params(new_params)** - This method changes the value of **algo_params** and of **algo_params_upon_instantiation** to that of **new_params**. Moreover:

  - The **policy**, **input_shape**, **output_shape** and **n_actions** variables are created: these are needed for instantiating the actual offline RL algorithm from MushroomRL.

    The **policy** is an epsilon greedy policy, while the other variables are created according to the choice of **regressor_type**. To see how read the web page linked in the explanation of the method **pre_learn_check()** of the Class *ModelGeneration*.
  - A structured dictionary is created: this is needed in order to properly instantiate a MushroomRL **agent** object. Moreover in order to instantiate such an object we also need to extract the **current_actual_value** from the **new_params** dictionary, which is made of *HyperParameter* objects.
  - An object of Class *mushroom_rl.core.Agent* (based on the value of the member **model**) is created passing to it the structured dictionary created in the previous step. This is saved in the member **algo_object**.

  Parameters:

  - **new_params** (*dict*) - This must be a flat dictionary containing all the parameters needed for this block to work.

Returns:

- (*bool*) - This is *True* if **new_params** was set successfully, else it is *False*.

---

16 model_generation_default.py

In the module *model_generation_default.py* the dictionary *automatic_model_generation_default* is present.

This is used as default setting in the Block *AutoModelGeneration* when the user does not want to specify anything. The default setting is given by:

- For the offline RL case when an environment is present we use a block of Class *ModelGenerationMushroomOfflineFQI* with default hyper-parameters, evaluated using the metric Class *DiscountedReward* for 10 episodes, tuned with the tuner Class *TunerGenetic*. The input loader Class is *LoadUniformSubSampleWithReplacementAndEnv* with each dataset being 10000 steps long.
- For the offline RL case when an environment is not present we use a block of Class *ModelGenerationMushroomOfflineFQI* with default hyper-parameters, evaluated using the metric Class *TDError*, tuned with the tuner Class *TunerGenetic*. The input loader Class is *LoadUniformSubSampleWithReplacement* with each dataset being 10000 steps long.
- A block of Class *ModelGenerationMushroomOnlinePPO* with default hyper-parameters, evaluated using the metric Class *DiscountedReward* for 10 episodes, tuned with the tuner Class *TunerGenetic*. The input loader Class is *LoadSameEnv*.

---

17 model_generation_automatic.py

In the module *model_generation_automatic.py* the Class *AutoModelGeneration* is implemented.

17.1 AutoModelGeneration

This Class optimises over the proposed algorithms: it calls a Tuner on each proposed algorithm and picks the most performing one according to some metric.

The Class *AutoModelGeneration* inherits from the Class *ModelGeneration*.

Note that in this Class:

- **works_on_online_rl** is equal to *True*.
- **works_on_offline_rl** is equal to *True*.
- **works_on_box_action_space** is equal to *True*.
- **works_on_discrete_action_space** is equal to *True*.
- **works_on_box_observation_space** is equal to *True*.
- **works_on_discrete_observation_space** is equal to *True*.
- **pipeline_type** is equal to *'online'* if **works_on_online_rl** is equal to *True*, else it is *'offline'*.
- **is_parametrised** is equal to *False*, hence this block cannot go in a *Tuner* object. Indeed it makes no sense to have **is_parametrised** is equal to *True* because this block is already performing optimisation of the hyper-parameters.

Note that the members **works_on_online_rl**, **works_on_offline_rl**, **works_on_box_action_space**, **works_on_discrete_action_space**, **works_on_box_observation_space** and **works_on_discrete_observation_space** are *True* so that one can specify the right tuners for the problem at hand.

If the tuners provided in the **tuner_blocks_dict** are incompatible with the problem at hand then no tuner will be tuned and the method **learn()** of the *AutoModelGeneration* block will fail.

Initialiser:
__init__(eval_metric, obj_name, seeder=2, tuner_blocks_dict=None, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **tuner_blocks_dict** (*dict*, None) - This must be a dictionary where the key is a string while the value is an object of a Class inheriting from the Class *Tuner*.

  If *None* is provided then the default dictionary **automatic_model_generation_default** present in the module **model_generation_default.py** will be used.

Non-Parameters Members:

- **tuner_blocks_dict_upon_instantiation** (*dict*, None) - This is a deep copy of the original value of **tuner_blocks_dict**.

- **fully_instantiated** (*bool*, False) - This is set to *True* once the block is fully instantiated, that is when the method **full_block_instantiation()** has been called.
- **info_MDP** (mushroom_rl.environment.MDPInfo, None) - This is an object of a Class implemented in MushroomRL. It contains the observation space, the action space, gamma and the horizon of the environment.

Methods:

- **full_block_instantiation(info_MDP)** - This method always returns *True* and sets the member **info_MDP**.

  This method does not call the methods **full_block_instantiation()** of all blocks present in **tuner_blocks_dict** because the default of such member, setted via **automatic_model_generation_default**, may contain several blocks that do not work on the problem at hand: we just want to skip over such blocks. This is taken care of in the method **learn()**.

  Parameters:

  - **info_MDP** (*mushroom_rl.environment.MDPInfo*) - This must be an object of Class containing the observation space, the action space, gamma and the horizon of the environment. This is needed for instantiating an object of Class *mushroom_rl.core.Agent*.

  Returns:

  - (*bool*) - This method always returns *True*.

- **pre_learn_check(train_data=None, env=None)** - This method simply returns *True* and updates the value of **tuner_blocks_dict** to **tuner_blocks_dict_upon_instantiation**. This is needed for re-loading objects.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - (*bool*) - This method overrides the one of the *base* Class *ModelGeneration* and it always returns *True*. Why is this needed? Because there is only one default dictionary for **tuner_blocks_dict** therefore it contains a variety of blocks.

    We do not want to stop learning an automatic block only because it contains a block that only works on *Box* observation spaces, while the problem has *Discrete* observation space: we just want to skip over it.

- **learn(train_data=None, env=None)** - This method calls the learn method **learn()** implemented in the Class *ModelGeneration*.

  Before calling the method **tune()** of a tuner:

  - We assign the **pipeline_type** to the **block_to_opt** present in the tuner.
  - We call the method **pre_learn_check()** of the **block_to_opt** present in the tuner. If this returns *False* then the currently selected tuner will be skipped.
  - If the call to the method **pre_learn_check()** of the **block_to_opt** present in the tuner, was successful then we call the method **full_block_instantiation()** of the **block_to_opt** present in the tuner.

    If this returns *False* then the currently selected tuner will be skipped.

Note that we skip over tuners that have input loader and metric not consistent with each other: this is checked by calling the method **is_metric_consistent_with_input_loader()** of the tuner.

After having called the method **tune()** of a tuner if everything was successful we update the local variables **best_agent** and **best_agent_eval**.

If after having gone through each tuner at least one was successfully tuned then the best agent found overall will be learnt over the original starting input given to the *AutoModelGeneration* block, and the corresponding object of Class *BlockOutput* is returned.

Otherwise if no tuner was tuned successfully, out of the ones present in the **tuner_blocks_dict** an empty object of Class *BlockOutput* is returned.

Parameters:

- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- (*BlockOutput*) - If this block was learnt successfully this object contains: **train_data**, **env**, **policy**, **policy_eval**. Else this object will have *None* values for the members: **train_data**, **env**, **policy**, **policy_eval**.

• **get_params()** - This method returns a deep copy of **tuner_blocks_dict**.

Returns:

- (*dict*) - This is a deep copy of **tuner_blocks_dict**.

• **set_params(new_params_dict)** - This method replaces the current **tuner_blocks_dict** and updates **tuner_blocks_dict_upon_instantiation** to **new_params_dict**.

Parameters:

- **new_params_dict** (*dict*) - This must be a flat dictionary for the parameters of all model generation blocks present in this automatic block: it replaces the current **tuner_blocks_dict**.

Returns:

- (*bool*) - This is *True* if **new_params_dict** was set successfully, else it is *False*.

• **analyse()** - This method is not yet implemented. It should analyse the learnt *AutoModelGeneration* block, namely it should evaluate it according to the provided **eval_metric**.

• **update_verbosity(new_verbosity)** - This method calls the *base* method **update_verbosity()** implemented in the Class *Block*. Then for all *Tuner* object in the **tuner_blocks_dict** it calls the corresponding method **update_verbosity()**.

Parameters:

- **new_verbosity** (int) - This must be a positive integer representing the new **verbosity**.

---

18 block_output.py

In the module *block_output.py* the Class *BlockOutput* is implemented.

18.1 BlockOutput

This Class is used to represent the output of a generic block of a Class inheriting from the Class *Block*.

The Class *BlockOutput* inherits from the Class *AbstractUnit*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', train_data=None, env=None, policy=None, policy_eval=None)

Parameters:

- **train_data** (*BaseDataSet*, None) - If specified this must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - If specified this must be an object of a Class inheriting from the Class *BaseEnvironment*.
- **policy** (*BasePolicy*, None) - If specified this must be an object of a Class inheriting from the Class *BasePolicy*.
- **policy_eval** (*float*, None) - If specified this must represent the evaluation, or KPI, of the **policy**. This is a dictionary containing the mean of the evaluation and the variance of the evaluation.

  Note that the Classes *Metric* that are used with *ModelGeneration* blocks must have two members: **eval_mean** and **eval_var**. These are used in the method **learn()** of the Classes *OfflineRLPipeline* and *OnlineRLPipeline* to construct the **policy_eval**.

Non-Parameters Members:

- **n_outputs** (*int*) - This is a positive integer and it represents the number of actual outputs that a block wants to save in an object of this Class.

Methods:

- **make_policy_deterministic()** - This method renders the policy deterministic: the approximator is extracted and added to the member **approximator** of the **policy** and the member **policy** of the **policy** is turned into an object of Class *mushroom_rl.policy.DeterministicPolicy*.

  Note that this means that you can call batch evaluation in the metric *DiscountedReward*: to see more about what this means see the Class *DiscountedReward*.

---

19 dataset.py

In the module *dataset.py* the Classes *BaseDataSet* and *TabularDataSet* are implemented.

19.1 BaseDataSet

The Class *BaseDataSet* is an abstract Class used as base class for all types of data one can have: tabular data, image data, text data.

The Class *BaseDataSet* is an abstract Class, and so it inherits from *ABC*, but it also inherits from the Class *AbstractUnit*.

Initialiser:
__init__(observation_space, action_space, discrete_actions, discrete_observations, gamma, horizon, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **observation_space** (*Box*/*Discrete*) - This is an object of either Class *Box* or Class *Discrete*: if the **observation_space** is discrete then it will be an object inheriting from the Class *Discrete*, else it will be an object inheriting from the Class *Box*.

  Both of these Classes are implemented in the *spaces.py* module in MushroomRL.
  cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/utils/spaces.py

  Since the RL algorithms available in this library are wrappers of those implemented in MushroomRL, and since in MushroomRL, as of now, only *Box* and *Discrete* spaces are supported, then also in this library we only support *Box* and *Discrete* spaces.

  For example *MultiDiscrete* or *Dict* spaces are not currently supported.

- **action_space** (*Box*/*Discrete*) - This is an object of either Class *Box* or Class *Discrete*: if the **action_space** is discrete then it will be an object inheriting from the Class *Discrete*, else it will be an object inheriting from the Class *Box*.

  Both of these Classes are implemented in the *spaces.py* module in MushroomRL.
  cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/utils/spaces.py

  Since the RL algorithms available in this library are wrappers of those implemented in MushroomRL, and since in MushroomRL, as of now, only *Box* and *Discrete* spaces are supported, then also in this library we only support *Box* and *Discrete* spaces.

  For example *MultiDiscrete* or *Dict* spaces are not currently supported.

- **discrete_actions** (*bool*) - This is *True* if the environment has discrete actions. Else it is *False*.

- **discrete_observations** (*bool*) - This is *True* if the environment has discrete observations. Else it is *False*.

- **gamma** (*float*) - This is a real value in [0, 1] and it represents the discount factor of the MDP.

- **horizon** (*int*) - This is a positive integer and it represents the horizon of the MDP.

Methods:

- **info()** - This method is a *property* and it is used for compatibility with MushroomRL algorithms. This method returns an object inheriting from the Class *MDPInfo*, which is a Class implemented in MushroomRL in the module *environment.py*.
  cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/core/environment.py

  Returns:

  – *(mushroom_rl.environment.MDPInfo)* - This object contains the **observation_space**, the **action_space,** the discount factor **gamma** and the **horizon**.

19.2 TabularDataSet

The Class *TabularDataSet* is used as base class for all tabular data: it has a member containing tabular data.

The Class *TabularDataSet* inherits from the Class *BaseDataSet*.

Initialiser:
__init__(dataset, observation_space, action_space, discrete_actions, discrete_observations, gamma, horizon, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **dataset** (*list*) - This must be a list where each entry of the list is made up of: the current state, the drawn action, the reward, the next state, the absorbing state flag and the episode terminal flag.

  This is needed in order to use MushroomRL algorithms.

Methods:

- **tuples_to_lists()** - This method transforms a list of tuples into a list of lists. This is needed since tuples are immutable. This method is called in the **Initialiser** only when the **dataset** parameter is not *None*.

- **arrays_as_data(states, actions, rewards, next_states, absorbings, lasts)** - This method constructs a dataset that can be passed to MushroomRL offline agents. Specifically it constructs a list of lists in which each list contains: the current state, the current action, the current reward, the next state, the absorbing state flag and the episode terminal flag.

  This method simply calls the method **mushroom_rl.utils.dataset.arrays_as_dataset()**.

  This is a *static method*.

  Parameters:

  - **states** (*numpy.array*) - This must be a *numpy.array* containing the states. Each state refers to a single time step.
  - **actions** (*numpy.array*) - This must be a *numpy.array* containing the actions. Each action refers to a single time step.
  - **rewards** (*numpy.array*) - This must be a *numpy.array* containing the rewards. Each reward refers to a single time step.
  - **next_states** (*numpy.array*) - This must be a *numpy.array* containing the next state the agent reaches by taking the sampled action in the current state.
  - **absorbings** (*numpy.array*) - This must be a *numpy.array* containing the flags indicating absorbing states.
  - **lasts** (*numpy.array*) - This must be a *numpy.array* containing the flags indicating end of episodes states.

  Returns:

  - **created_dataset** (*list*) - This is a list of lists constructed as described above.

- **parse_data()** - This method simply calls the method **mushroom_rl.utils.dataset.parse_dataset()**.

  Returns:

- (*numpy.array*, *numpy.array*, *numpy.array*, *numpy.array*, *numpy.array*, *numpy.array*) - The six *numpy.array* that are obtained by parsing the dataset: states, actions, rewards, next_states, absorbing state flags, episode terminals flags.

- **get_states()** - This method calls the method **parse_data()** and selects the first *numpy.array*.

  Returns:

  - (*numpy.array*) - The array of current states.

- **get_actions()** - This method calls the method **parse_data()** and selects the second *numpy.array*.

  Returns:

  - (*numpy.array*) - The array of actions.

- **get_rewards()** - This method calls the method **parse_data()** and selects the third *numpy.array*.

  Returns:

  - (*numpy.array*) - The array of rewards.

- **get_next_states()** - This method calls the method **parse_data()** and selects the fourth *numpy.array*.

  Returns:

  - (*numpy.array*) - The array of the next states.

- **get_absorbing()** - This method calls the method **parse_data()** and selects the fifth *numpy.array*.

  Returns:

  - (*numpy.array*) - The array of absorbing states flags.

- **get_episode_terminals()** - This method calls the method **parse_data()** and selects the sixth *numpy.array*.

  Returns:

  - (*numpy.array*) - The array of episode terminal states flags.

---

20 environment.py

In the module *environment.py* the Classes *BaseEnvironment*, *BaseWrapper*, *BaseObservationWrapper*, *BaseActionWrapper*, *BaseRewardWrapper* are implemented.

Wrappers of actual environments that can be used are also implemented: *BaseGridWorld*, *BaseCarOnHill*, *BaseCartPole*, *BaseInvertedPendulum*, *LQG*, *BaseMujoco*, *BaseHalfCheetah*, *BaseAnt*, *BaseHopper*, *BaseHumanoid*, *BaseSwimmer*, *BaseWalker2d*.

20.1 BaseEnvironment

This is the base environment Class based on the OpenAI Gym class. Part of this class is a re-adaptation of code copied from:

- OpenAI gym: cf. https://github.com/openai/gym/blob/master/gym/core.py
- MushroomRL: cf. https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/core/environment.py

This can be sub-classed by the user to create their own specific environment. You must sub-class from this Class in order to use this library.

You must use *Box* and *Discrete* environment spaces from *MushroomRL*.

The Class *BaseEnvironment* is an abstract Class, and so it inherits from *ABC*, but it also inherits from the Class *AbstractUnit*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **observation_space** (*Box*/*Discrete*, None) - This must be a space from *MushroomRL*, either *Box* or *Discrete*: it must be an object inheriting from one of following two the Classes: *mushroom_rl.utils.spaces.Box* or *mushroom_rl.utils.spaces.Discrete*.
- **action_space** (*Box*/*Discrete*, None) - This must be a space from *MushroomRL*, either *Box* or *Discrete*.
- **gamma** (*float*, None) - This is the value of the gamma of the MDP.
- **horizon** (*int*, None) - This is the horizon of the MDP.

Methods:

- **step(action)** - This method is an abstract method and it must be implemented in a sub-Class of this Class.

  Parameters:

  - **action** (*object*) - This is an action to be applied on the environment.

  Returns:

  - **observation** (*object*) - This is the current observation returned to the agent by the environment.

  - **reward** (*float*) - This is the reward obtained by calling the previous action.
  - **done** (*bool*) - This is *True* if an episode has ended, else it is *False*.
  - **info** (*dict*) - This dictionary contains auxiliary diagnostic information about the environment.

- **reset(state=None)** - This method is an abstract method and it must be implemented in a sub-Class of this Class.

  Note that all methods **reset** in all environments must have this signature otherwise MushroomRL RL algorithms are going to fail!

- **render(mode='human')** - This method is an abstract method and it must be implemented in a sub-Class of this Class.

- **close()** - If needed, this method can be implemented in a sub-Class of this Class.

- **seed(seed=None)** - If needed, this method can be implemented in a sub-Class of this Class.

- **stop()** - This method is used to stop an MDP. This is needed for backward compatibility with *MushroomRL*.

  If needed, this method can be implemented in a sub-Class of this Class.

- **unwrapped()** - This method completely unwraps the environment contained in the Class.

  Returns:

  - (*BaseEnvironment*) - This is the base non-wrapped environment instance.

- **info()** - This method is a property method and it is used to extract information about the environment.

  Returns:

  - (*mushroom_rl.environment.MDPInfo*) - This method returns an object of Class *mushroom_rl.environment.MDPInfo*: it contains the observation space, the action space, gamma and the horizon of the environment.

- **_sample_from_box(space)** - This method generates a single random sample inside of the *Box* space. It can only be called when the space is a *Box* (i.e: continuous).

  This method was copied from OpenAI gym: cf. https://github.com/openai/gym/blob/master/gym/spaces/box.py

  Parameters:

  - **space** (*Box*) - This must be a *Box* space from *MushroomRL*. This is the space from which to sample from.

  Returns:

  - **sample** (*numpy.array*) - This is a random sample from the *Box* space.

- **sample_from_box_action_space()** - This method generates a single random sample inside of the *Box* action space. It can only be called when the action space is a *Box* (i.e: continuous). This method simply calls the method **_sample_from_box()** passing to it the **action_space**.

  Returns:

  - **sample** (*numpy.array*) - This is a random sample from the *Box* action space.

- **sample_from_box_observation_space()** - This method generates a single random sample inside of the *Box* observation space. It can only be called when the observation space is a *Box* (i.e: continuous). This method simply calls the method **_sample_from_box()** passing to it the **observation_space**.

  Returns:

  - **sample** (*numpy.array*) - This is a random sample from the *Box* observation space.

- **set_params(params_dict)** - This method is used to set the parameters of an environment. By passing a dictionary containing as keys the correct string member name of an environment it can be used to set a value to such members, according to the value provided in the **params_dict**.

  Parameters:

  - **params_dict** (*dict*) - This is a dictionary containing as keys the correct string member name for which we want to set a new value, and as value the new value to which we want to set the members.

- **get_params(params_names)** - This method is used to obtain the value for certain members of an environment. It generates a dictionary with keys the same keys provided as input and as values the current value of the corresponding members.

  Parameters:

  - **params_names** (*list*) - This is a list containing, as string, the name of the members for which we want to retrieve their current value.

  Returns:

  - **params_dict** (*dict*) - This is a dictionary containing as keys the correct string member name and as value the corresponding member current value.

20.2 BaseWrapper

This is the base wrapper Class based on the OpenAI Wrapper Class. Part of this class is a re-adaptation of code copied from OpenAI gym:
cf. https://github.com/openai/gym/blob/master/gym/core.py

This is used as base Class for observation wrappers, action wrappers and reward wrappers.

Other kind of wrappers can be created inheriting from this Class.

The Class *BaseWrapper* inherits from the Class *BaseEnvironment*. This is an *Abstract Class*.

Initialiser:
__init__(env, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **env** (*BaseEnvironment*) - This is the environment that needs to be wrapped. It must be an object of a Class inheriting from the Class *BaseEnvironment*.

Non-Parameters Members:

- **observation_space** (*Box*/*Discrete*) - This is set equal to the value of **observation_space** of the **env**.
- **action_space** (*Box*/*Discrete*, None) - This is set equal to the value of **action_space** of the **env**.
- **gamma** (*float*) - This is set equal to the value of **gamma** of the **env**.
- **horizon** (*int*) - This is set equal to the value of **horizon** of the **env**.

Methods:

- **unwrapped()** - This method simply calls the method **unwrapped()** of the **env**.

  Returns:

  - (*BaseEnvironment*) - The base non-wrapped environment instance.
- **stop()** - This method simply calls the method **stop()** of the **env**.
- **step(action)** - This method simply calls the method **step()** of the **env**.
- **reset(state=None)** - This method simply calls the method **reset()** of the **env**.
- **render(mode='human')** - This method simply calls the method **render()** of the **env**.
- **close()** - This method simply calls the method **close()** of the **env**.
- **seed(seed=None)** - This method simply calls the method **seed()** of the **env**.

20.3 BaseObservationWrapper

To create an observation wrapper you must create a new Class inheriting from this Class. You must override the method **observation()** and in the **Initialiser** you must:

- Call the **Initialiser** of *BaseWrapper* via: *super().__init__(env)*
- Properly modify the observation space.

Part of this Class is a re-adaptation of code copied from OpenAI gym:
cf. https://github.com/openai/gym/blob/master/gym/core.py

The Class *BaseObservationWrapper* inherits from the Class *BaseWrapper*. This is an *Abstract Class*.

Initialiser:
__init__(env, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **reset(state=None)** - This method calls the method **observation()**, that must be implemented in a sub-Class of this Class, onto the output of the method **reset()** of the **env**.
- **step(action)** - This method calls the method **observation()**, that must be implemented in a sub-Class of this Class, onto the observation that is part of the output of the method **step()** of the **env**.
- **observation(observation)** - This method is an abstract method and it must be implemented in a sub-Class of this Class. By implementing this method an observation wrapper is effectively constructed.

20.4 BaseActionWrapper

To create an action wrapper you must create a new Class inheriting from this Class. You must override the method **action()** and in the **Initialiser** you must:

- Call the **Initialiser** of *BaseWrapper* via: *super().__init__(env)*
- Properly modify the action space.

Part of this Class is a re-adaptation of code copied from OpenAI gym:
cf. https://github.com/openai/gym/blob/master/gym/core.py

The Class *BaseActionWrapper* inherits from the Class *BaseWrapper*. This is an *Abstract Class*.

Initialiser:
__init__(env, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **step(action)** - This method calls the method **step()** of the **env**, on the output of the method **action()** that must be implemented in a sub-Class of this Class.
- **action(action)** - This method is an abstract method and it must be implemented in a sub-Class of this Class. By implementing this method an action wrapper is effectively constructed.

20.5 BaseRewardWrapper

To create a reward wrapper you must create a new Class inheriting from this Class. You must override the method **reward()** and in the **Initialiser** you must:

- Call the **Initialiser** of *BaseWrapper* via: *super().__init__(env)*

Part of this Class is a re-adaptation of code copied from OpenAI gym:
cf. https://github.com/openai/gym/blob/master/gym/core.py

The Class *BaseRewardWrapper* inherits from the Class *BaseWrapper*. This is an *Abstract Class*.

Initialiser:
__init__(env, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Methods:

- **step(action)** - This method calls the method **step()** of the **env**. Then the reward contained in the output of such call is modified by calling on it the method **reward()**, that must be implemented in a sub-Class of this Class.
- **reward(reward)** - This method is an abstract method and it must be implemented in a sub-Class of this Class. By implementing this method a reward wrapper is effectively constructed.

20.6 BaseGridWorld

This Class wraps the GridWorld Class from MushroomRL: this is needed for the correct working of this library.

The Class *BaseGridWorld* inherits from the Class *BaseEnvironment* and from the MushroomRL Class *GridWorld*.

Initialiser:
__init__(height, width, goal, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', start=(0,0))

To see the specifics of this Class see the documentation about the Class *BaseEnvironment* and the documentation of *MushroomRL* about the Class *GridWorld*.
cf. `https://mushroomrl.readthedocs.io/en/latest/source/mushroom_rl.environments.html#module-mushroom_rl.environments.grid_world`

20.7 BaseCarOnHill

This Class wraps the CarOnHill Class from MushroomRL: this is needed for the correct working of this library.

The Class *BaseCarOnHill* inherits from the Class *BaseEnvironment* and from the MushroomRL Class *CarOnHill*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', horizon=100, gamma=0.95)

To see the specifics of this Class see the documentation about the Class *BaseEnvironment* and the documentation of *MushroomRL* about the Class *CarOnHill*.
cf. https://mushroomrl.readthedocs.io/en/latest/source/mushroom_rl.environments.html#module-mushroom_rl.environments.car_on_hill

20.8 BaseCartPole

This Class wraps the CartPole Class from MushroomRL: this is needed for the correct working of this library.

The Class *BaseCartPole* inherits from the Class *BaseEnvironment* and from the MushroomRL Class *CartPole*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', m=2, M=8, l=0.5, g=9.8, mu=1e-2, max_u=50, noise_u=10, horizon=3000, gamma=0.95)

To see the specifics of this Class see the documentation about the Class *BaseEnvironment* and the documentation of *MushroomRL* about the Class *CartPole*.
cf. https://mushroomrl.readthedocs.io/en/latest/source/mushroom_rl.environments.html#module-mushroom_rl.environments.cart_pole

20.9 BaseInvertedPendulum

This Class wraps the InvertedPendulum Class from MushroomRL: this is needed for the correct working of this library.

The Class *BaseInvertedPendulum* inherits from the Class *BaseEnvironment* and from the MushroomRL Class *InvertedPendulum*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', m=1, l=1, g=9.8, mu=1e-2, max_u=5, horizon=5000, gamma=0.99)

To see the specifics of this Class see the documentation about the Class *BaseEnvironment* and the documentation of *MushroomRL* about the Class *InvertedPendulum*.
cf. `https://mushroomrl.readthedocs.io/en/latest/source/mushroom_rl.environments.html#module-mushroom_rl.environments.inverted_pendulum`

20.10 LQG

This Class implements an LQG environment (i.e: a Linear-Quadratic Gaussian control (LQG) problem). The code for this Class is based on the following code:
cf. https://github.com/T3p/potion/blob/master/potion/envs/lq.py

The Class *LQG* inherits from the Class *BaseEnvironment*.

Initialiser:
__init__(obj_name, A=np.eye(1), B=np.eye(1), Q=np.eye(1), R=np.eye(1), max_pos=1.0, max_action=1.0, env_noise=np.eye(1), controller_noise=np.eye(1), horizon=10, gamma=0.9, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **A** (*numpy.ndarray*, np.eye(1)) - This is the state dynamics matrix.
- **B** (*numpy.ndarray*, np.eye(1)) - This is the action dynamics matrix.
- **Q** (*numpy.ndarray*, np.eye(1)) - This is the cost weight matrix for the state. It must be a positive-definite matrix (to always have a negative reward).
- **R** (*numpy.ndarray*, np.eye(1)) - This is the cost weight matrix for the action. It must be a positive-definite matrix (to always have a negative reward).
- **max_pos** (*float*, 1.0) - This is the maximum value that the state can reach.
- **max_action** (*float*, 1.0) -This is the maximum value that the action can reach.
- **env_noise** (*numpy.ndarray*, np.eye(1)) - This is the covariance matrix representing the Gaussian environment noise.
- **controller_noise** (*numpy.ndarray*, np.eye(1)) - This is the covariance matrix representing the Gaussian controller noise.
- **horizon** (*int*, 10) - This is the horizon of the MDP.
- **gamma** (*float*, 0.9) - This is the discount factor of the MDP.

Non-Parameters Members:

- **is_eval_phase** (*bool*, False) - This is *True* if the environment is used for evaluating a policy: what happens is that the **controller_noise** is added to the action selected by the policy, and then fed to the simulator.

  Otherwise it is False.

  This is used to represent the fact that even if we have learnt a theoretically optimal policy, in practice to execute it there is going to be some noise and so the resulting action taken in the real world will be different from the one selected by the policy.

  This parameter can be set automatically by the evaluation metric.

Methods:

- **step(action)** - This method is used to run one step of the environment dynamics. As always it returns observation, reward, done and info. For more information see the documentation of the Class *BaseEnvironment*.

- **reset(state=None)** - This method is used to reset the environment: by default, random uniform initialisation. If the **state** is not *None* then the provided **state** is used for re-setting the environment.

  Parameters:

  - **state** (*numpy.array*, None) - This is a vector representing the new state for the environment.

  Returns:

  - **state** (*numpy.array*, None) - This is a vector representing the new state for the environment.

- **seed(seed=None)** - This method is used to seed the environment. This method calls the method **set_local_prng()** passing to it the provided **seed**.

  Parameters:

  - **seed** (*int*, None) - This is a positive integer representing the new seed to use.

- **render(mode='human', close=False)** - This method is used to render the environment.

- **get_optimal_K()** - This method computes the optimal gain K and returns the optimal policy given by: $-K * s$ where s is the state. To do so the discrete ARE is solved using *scipy*.

  Returns:

  - **-K** (*numpy.ndarray*) - This is the optimal gain matrix.

20.11 BaseMujoco

This Class wraps the Mujoco environments: every Mujoco environment inherits from this Class.

The Class *BaseMujoco* inherits from the Class *BaseEnvironment*. This is an *Abstract* Class.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **mujoco_env** (*gym.envs.mujoco.mujoco_env.MujocoEnv*, None) - This is a Mujoco environment instance: an object of a Class inheriting from OpenAI gym Class: *gym.envs.mujoco.mujoco_env.MujocoEnv*.

  This is set in sub-Classes of this Class.

- **n_steps** (*int*, 0) - This is a counter for the horizon of the environment. It is the number of total steps that have happened so far.

Methods:

- **_update_counter(out_step)** - This method keeps track of how many steps the agent has interacted with the environment for, and if needed it sets **done** equal to *True* in the output obtained from the call of the method **step()** of the Mujoco environment (i.e: an object of a Class inheriting from the Class *gym.envs.mujoco.mujoco_env.MujocoEnv*).

  Parameters:

  - **out_step** (*tuple*) - This is the output from the call to the method **step()** of the Mujoco environment (i.e: an object of a Class inheriting from the Class *gym.envs.mujoco.mujoco_env.MujocoEnv*).

  Returns:

  - *tuple(out_step)* (*tuple*) - This is the new output of the call to the method **step()** of a Class inheriting from this Class: in case **n_steps** reached **horizon** we set **done** equal to *True*.

- **step(action)** - This method calls the method **step()** of the object **mujoco_env** and then it calls the method **_update_counter()**.

  Parameters:

  - **action** (*object*) - This is an action to be applied on the environment.

  Returns:

  - **out** (*tuple*) - This is the usual tuple containing observation, reward done and info.

- **reset(state=None)** - This method calls the method **reset()** of **mujoco_env** and it sets **n_steps** to 0. It returns the observation that is where the environment starts.

  The signature contains a useless **state=None** since this is needed for compatibility with MushroomRL RL algorithms.

- **seed(seed=None)** - This method simply calls the method **seed()** of the **mujoco_env**.

- **render(mode='human')** - This method calls the method **render()** of the **mujoco_env**.

- **set_local_prng(new_seeder)** - This method overrides the method implemented in the Class *AbstractUnit*. This method calls the method **seed()** of this Class, passing to it the **new_seeder**. This method is used to adjust to the fact that OpenAI does not use the system of using the **local_prng**, defined in the Class *AbstractUnit*, but have their own way of doing it.

  Without this method by calling the original method **set_local_prng()** implemented in the Class *AbstractUnit* I would have no effect on the environment: the method **set_local_prng()** is called in the Classes *Metric* and in some of the *DataGeneration* Classes so this method needs to work properly.

  To learn more about this see the implementation of the method **set_local_prng()** implemented in the Class *AbstractUnit* or see the chapter **On environments** in the section **Further Details**.

  Parameters:

  - **new_seeder** (**int**) - This is a non-negative integer and it represents the new seeder to be used for creating a new local PRNG.

- **_initialise_mdp_properties(gamma, horizon)** - This method sets the MDP properties: the discount factor and the horizon. Moreover it converts the spaces contained in **mujoco_env** to use those implemented in MushroomRL.

  Indeed all environments must use the spaces *Box* and *Discrete* implemented in MushroomRL, otherwise it will not be possible to use the RL algorithms in MushroomRL.

  Parameters:

  - **gamma** (*float*) - This is the discount factor of the MDP.
  - **horizon** (*int*) - This is the horizon of the MDP.

To see some more specifics of this Class see the documentation about the Class *gym.envs.mujoco.mujoco_env.MujocoEnv*. cf. https://github.com/openai/gym/blob/master/gym/envs/mujoco/mujoco_env.py

20.12 BaseHalfCheetah

This Class wraps the HalfCheetahEnv Mujoco environment.

The Class *BaseHalfCheetah* inherits from the Class *BaseMujoco*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', gamma=0.99, horizon=1000, xml_file='half_cheetah.xml', forward_reward_weight=1.0, ctrl_cost_weight=0.1, reset_noise_scale=0.1, exclude_current_positions_from_observation=True)

Parameters:

- **gamma** (*float*, 0.99) - This is the discount factor of the MDP.
- **horizon** (*int*, 1000) - This is the horizon of the MDP.

Note that in this Class **mujoco_env** is an instance of *gym.envs.mujoco.half_cheetah_v3.HalfCheetahEnv*.

To see some more specifics of this Class see the documentation about the Mujoco Class *HalfCheetahEnv*.
cf. https://github.com/openai/gym/blob/master/gym/envs/mujoco/half_cheetah_v3.py

20.13 BaseAnt

This Class wraps the AntEnv Mujoco environment.

The Class *BaseAnt* inherits from the Class *BaseMujoco*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', gamma=0.99, horizon=1000, xml_file='ant.xml', ctrl_cost_weight=0.5, contact_cost_weight=5e-4, healthy_reward=1.0, terminate_when_unhealthy=True, healthy_z_range=(0.2, 1.0), contact_force_range=(-1.0, 1.0), reset_noise_scale=0.1, exclude_current_positions_from_observation=True)

Parameters:

- **gamma** (*float*, 0.99) - This is the discount factor of the MDP.
- **horizon** (*int*, 1000) - This is the horizon of the MDP.

Note that in this Class **mujoco_env** is an instance of *gym.envs.mujoco.ant_v3.AntEnv*.

To see some more specifics of this Class see the documentation about the Mujoco Class *AntEnv*.
cf. https://github.com/openai/gym/blob/master/gym/envs/mujoco/ant_v3.py

20.14 BaseHopper

This Class wraps the HopperEnv Mujoco environment.

The Class *BaseHopper* inherits from the Class *BaseMujoco*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', gamma=0.99, horizon=1000, xml_file='hopper.xml', forward_reward_weight=1.0, ctrl_cost_weight=1e-3, healthy_reward=1.0, terminate_when_unhealthy=True, healthy_state_range=(-100.0, 100.0), healthy_z_range=(0.7, float('inf')), healthy_angle_range=(-0.2, 0.2), reset_noise_scale=5e-3, exclude_current_positions_from_observation=True)

Parameters:

- **gamma** (*float*, 0.99) - This is the discount factor of the MDP.
- **horizon** (*int*, 1000) - This is the horizon of the MDP.

Note that in this Class **mujoco_env** is an instance of *gym.envs.mujoco.hopper_v3.HopperEnv*.

To see some more specifics of this Class see the documentation about the Mujoco Class *HopperEnv*.
cf. https://github.com/openai/gym/blob/master/gym/envs/mujoco/hopper_v3.py

20.15 BaseHumanoid

This Class wraps the HumanoidEnv Mujoco environment.

The Class *BaseHumanoid* inherits from the Class *BaseMujoco*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', gamma=0.99, horizon=1000, xml_file='humanoid.xml', forward_reward_weight=1.25, ctrl_cost_weight=0.1, contact_cost_weight=5e-7, contact_cost_range=(-np.inf, 10.0), healthy_reward=5.0, terminate_when_unhealthy=True, healthy_z_range=(1.0, 2.0), reset_noise_scale=1e-2, exclude_current_positions_from_observation=True)

Parameters:

- **gamma** (*float*, 0.99) - This is the discount factor of the MDP.
- **horizon** (*int*, 1000) - This is the horizon of the MDP.

Note that in this Class **mujoco_env** is an instance of *gym.envs.mujoco.humanoid_v3.HumanoidEnv*.

To see some more specifics of this Class see the documentation about the Mujoco Class *HumanoidEnv*.
cf. https://github.com/openai/gym/blob/master/gym/envs/mujoco/humanoid_v3.py

20.16 BaseSwimmer

This Class wraps the SwimmerEnv Mujoco environment.

The Class *BaseSwimmer* inherits from the Class *BaseMujoco*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', gamma=0.99, horizon=1000, xml_file="swimmer.xml", forward_reward_weight=1.0, ctrl_cost_weight=1e-4, reset_noise_scale=0.1, exclude_current_positions_from_observation=True)

Parameters:

- **gamma** (*float*, 0.99) - This is the discount factor of the MDP.
- **horizon** (*int*, 1000) - This is the horizon of the MDP.

Note that in this Class **mujoco_env** is an instance of *gym.envs.mujoco.swimmer_v3.SwimmerEnv*.

To see some more specifics of this Class see the documentation about the Mujoco Class *SwimmerEnv*.
cf. https://github.com/openai/gym/blob/master/gym/envs/mujoco/swimmer_v3.py

20.17 BaseWalker2d

This Class wraps the Walker2dEnv Mujoco environment.

The Class *BaseWalker2d* inherits from the Class *BaseMujoco*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', gamma=0.99, horizon=1000, xml_file="walker2d.xml", forward_reward_weight=1.0, ctrl_cost_weight=1e-3, healthy_reward=1.0, terminate_when_unhealthy=True, healthy_z_range=(0.8, 2.0), healthy_angle_range=(-1.0, 1.0), reset_noise_scale=5e-3, exclude_current_positions_from_observation=True)

Parameters:

- **gamma** (*float*, 0.99) - This is the discount factor of the MDP.
- **horizon** (*int*, 1000) - This is the horizon of the MDP.

Note that in this Class **mujoco_env** is an instance of *gym.envs.mujoco.walker2d_v3.Walker2dEnv*.

To see some more specifics of this Class see the documentation about the Mujoco Class *Walker2dEnv*.
cf. https://github.com/openai/gym/blob/master/gym/envs/mujoco/walker2d_v3.py

21 policy.py

In the module *policy.py* the Class *BasePolicy* is implemented.

21.1 BasePolicy

This Class is used as container for the output of any *ModelGeneration* block. The Class *BasePolicy* inherits from the Class *AbstractUnit*.

Initialiser:
__init__(policy, regressor_type, obj_name, approximator=None, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **policy** (*mushroom_rl.policy.Policy*) - This is the **policy** used in the RL algorithm implemented in the *ModelGeneration* block.

  Note that the **policy** alternatively can also be an object of any Class that exposes the method **draw_action()** taking as parameter just a single state.

- **regressor_type** (*str*) - This is a string and it can either be: *'action_regressor'*, *'q_regressor'* or *'generic_regressor'*. This is used to specify which one of the 3 possible kind of regressor, made available by MushroomRL, has been picked.

  This is needed in the Metric Classes: depending on the **regressor_type** we handle the evaluation phase differently. To see an example of how **regressor_type** is used see the implementation of the Class *DiscountedReward*.

- **approximator** (*mushroom_rl.approximators.regressor.Regressor*, None) - This represents the **approximator** used in the **policy** in the RL algorithm implemented in the *ModelGeneration* block.

  This is optional since all the relevant information are contained already in the **policy**, however this is used to discriminate between different policies and it plays a role in which *Metric* can be applied to which **policy**. To learn more about this read the documentation about the Class *DiscountedRewardMetric*.

  Note that the **approximator** alternatively can also be an object of any Class that exposes the method **predict()** taking as parameter either a single sample, or multiple samples.

22 hyperparameter.py

In the module *hyperparamter.py* the Classes *HyperParameter*, *Numerical*, *Integer*, *Real* and *Categorical* are implemented.

22.1 HyperParameter

This Class is used as generic base Class for all hyper-parameters. These are used to ease the hyper-parameters tuning process.

The Class *HyperParameter* is an abstract Class, and so it inherits from *ABC*, but it also inherits from the Class *AbstractUnit*.

Initialiser:
__init__(hp_name, current_actual_value, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', to_mutate=False)

Parameters:

- **hp_name** (*str*) - This is a string with the exact name of the hyper-parameter described by this class.

  Why this since there is already **obj_name**? **obj_name** can be personalised while **hp_name** must match exactly the name of the parameter used in a specific block.

- **current_actual_value** (*object*) - This is the current actual value of the parameter. It can be anything.

- **to_mutate** (*bool*) - This is either *True* or *False*. It is *True* if we want the *Tuner* to tune this hyper-parameter. It is *False* otherwise.

Non-Parameters Members:

- **block_owner_flag** (*int*, None) - This is an integer and it is used as flag to mark that a certain hyper-parameter belongs to some specific block. This is needed for assigning the correct hyper-parameters in the method **set_params()** of the Class *RLPipeline*.

  What happens is that in the *Tuner* Class I extract the parameters of a block with the method **get_params()** of that block, I mutate the hyper-parameters, and then I call the method **set_params()** of that block.

  Now if that block is a pipeline it will be composed of multiple blocks. The member **block_owner_flag** allows the method **set_params()** of a pipeline block to set the right hyper-parameters in the right blocks.

  For more see the documentation (or the implementation) of the **set_params()** and **get_params()** methods of the Class *RLPipeline*.

22.2 Numerical

This Class is the base Class for all numerical (both floats and integers) hyper-parameters.

The Class *Numerical* inherits from the Class *HyperParameter*. This is an *Abstract Class*.

Initialiser:
__init__(hp_name, current_actual_value, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', range_of_values=None, to_mutate=False, type_of_mutation='perturbation')

Parameters:

- **range_of_values** (*list*, None) - This must be a list of two, containing the lower bound and the upper bound of the range of values that the hyper-parameter can assume.
- **type_of_mutation** (*str*, 'perturbation') - This must be a string and it can either be:
  - 'perturbation': In this case the next possible values, occurring as a result of a mutation, can only be those in the range: current_actual_value $\cdot$ $[0.8, 1.2]$

    Selecting 'perturbation' we have less dramatic changes in the hyper-parameters from one step to the next: we must be between 0.8 times, and 1.2 times, of the current value.
  - 'mutation': In this case the next possible values, occurring as a result of a mutation, can be in the entire range of the possible values of the hyper-parameter.

22.3 Real

This is the Class representing all Numerical and Real (i.e: Continuous) hyper-parameters.

The Class *Real* inherits from the Class *Numerical*.

Initialiser:
__init__(hp_name, current_actual_value, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', range_of_values=None, to_mutate=False, type_of_mutation='perturbation')

Methods:

- **mutate(first_mutation)** - This method mutates the current hyper-parameter value by sampling from a continuous uniform distribution. Note that:
  - If **type_of_mutation** is *'mutation'* the sampling distribution is defined over the **range_of_values**.
  - If **type_of_mutation** is *'perturbation"* the sampling is defined:
    - Over the **range_of_values**, if **first_mutation** is *True*.
    - Over the range current_actual_value $\cdot [0.8, 1.2]$, clipped to **range_of_values**, if **first_mutation** is *False*.

  This method directly changes the value of **current_actual_value**.

  Note that a call to this method is successful only when **to_mutate** is equal to *True*.

  Parameters:

  - **first_mutation** (*bool*) - This is *True* if this is the first time we are mutating the hyper-parameter, else it is *False*.

22.4 Integer

This is the Class representing all Numerical and Integer (i.e: Discrete) hyper-parameters.

The Class *Integer* inherits from the Class *Numerical*.

Initialiser:
__init__(hp_name, current_actual_value, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', range_of_values=None, to_mutate=False, type_of_mutation='perturbation')

Methods:

- **mutate(first_mutation)** - This method mutates the current hyper-parameter value by sampling from a discrete uniform distribution. Note that:
  - If **type_of_mutation** is *'mutation'* the sampling distribution is defined over the **range_of_values**.
  - If **type_of_mutation** is *'perturbation"* the sampling is defined:
    * Over the **range_of_values**, if **first_mutation** is *True*.
    * Over the range current_actual_value $\cdot [0.8, 1.2]$, clipped to **range_of_values**, if **first_mutation** is *False*.

      Since the hyper-parameter here is an integer we cast the range of the mutation to *int* to make sure that we get out an integer.

  This method directly changes the value of **current_actual_value**.

  Note that a call to this method is successful only when **to_mutate** is equal to *True*.

  Parameters:

  - **first_mutation** (*bool*) - This is *True* if this is the first time we are mutating the hyper-parameter, else it is *False*.

22.5 Categorical

This is the Class representing all Categorical hyper-parameters.

The Class *Categorical* inherits from the Class *HyperParameter*.

Initialiser:
__init__(hp_name, current_actual_value, obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', possible_values=None, to_mutate=False)

Parameters:

- **possible_values** (*list*, None) - This must be an exhaustive list containing all the possible values the hyper-parameter can assume.

Methods:

- **mutate(first_mutation)** - This method mutates the current hyper-parameter value by sampling a random value from the list of **possible_values**.

  This method directly changes the value of **current_actual_value**.

  Note that a call to this method is successful only when **to_mutate** is equal to *True*.

  Parameters:

  - **first_mutation** (*bool*) - This is *True* if this is the first time we are mutating the hyper-parameter, else it is *False*.

    In this Class this member is useless: it is just kept to have the same interface across all *HyperParameter* objects.

---

23 input_loader.py

In the module *input_loader.py* the Classes *InputLoader*, *LoadSameEnv*, *LoadSameTrainData*, *LoadUniformSubSampleWithReplacement*, *LoadUniformSubSampleWithReplacementAndEnv*, *LoadDifferentSizeForEachBlock* and *LoadDifferentSizeForEachBlockAndEnv* are implemented.

23.1 InputLoader

This Class is used as generic base class for all input loaders. These are used to generate the right input to the different blocks that are trained by the *Tuner*. These Classes are needed because the *Tuner* are block agnostic and so we need a way to generate the right input for whichever automatic block might have called the *Tuner*.

The Class *InputLoader* is an *Abstract Class*, and so it inherits from *ABC*, but it also inherits from the Class *AbstractUnit*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **returns_dataset** (*bool*, None) - This is *True* if the input loader returns a dataset, otherwise it is *False*.

  This is needed for checking the consistency of the *Metric* with an *InputLoader*: this check is done in the *Tuner* via the method **is_metric_consistent_with_input_loader()**.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

- **returns_env** (*bool*, None) - This is *True* if the input loader returns an environment, otherwise it is *False*.

  This is needed for checking the consistency of the *Metric* with an *InputLoader*: this check is done in the *Tuner* via the method **is_metric_consistent_with_input_loader()**.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

23.2 LoadSameEnv

This particular Class simply returns **n_inputs_to_load** times the copy of the input **env**: indeed for online model generation we just have an environment so we can keep using it, anyway the trajectory followed by the agents depends by the particular action taken by each agent.

The Class *LoadSameEnv* inherits from the Class *InputLoader*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **returns_dataset** (*bool*, False) - This is *False* since this input loader does not generate a dataset.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

- **returns_env** (*bool*, True) - This is *True* since this input loader generates multiple copies of the original environment.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

Methods:

- **get_input(blocks, n_inputs_to_load, train_data=None, env=None)** - This method generates a list containing a number of environments equal to **n_inputs_to_load**. Each environment is a deep copy of **env**.

  Parameters:

  - **blocks** (*list*) - This is a list containing the blocks for which we need to load the input. This is used only in some *InputLoader*. In this *InputLoader* it is not used.
  - **n_inputs_to_load** (*int*) - This is the number of environments that will be deep copied.
  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns: This method returns two terms:

  - *None* - This indicates that no datasets were generated by this *InputLoader*.
  - **copied_envs** (*list*) - This is a list containing deep copies of the original environment **env**.

23.3 LoadSameTrainData

This particular Class simply returns **n_inputs_to_load** times the copy of the input **train_data**.

The Class *LoadSameTrainData* inherits from the Class *InputLoader*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **returns_dataset** (*bool*, True) - This is *True* since this input loader does generate a list of datasets.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

- **returns_env** (*bool*, False) - This is *False* since this input loader does not generate an environment.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

Methods:

- **get_input(blocks, n_inputs_to_load, train_data=None, env=None)** - This method generates a list containing a number of datasets equal to **n_inputs_to_load**. Each dataset is a deep copy of the original **train_data**.

  Parameters:

  - **blocks** (*list*) - This is a list containing the blocks for which we need to load the input. This is used only in some *InputLoader*. In this *InputLoader* it is not used.
  - **n_inputs_to_load** (*int*) - This is the number of environments that will be deep copied.
  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns: This method returns two terms:

  - **copied_train_data** (*list*) - This is a list containing deep copies of the original **train_data**.
  - *None* - This indicates that no environments were generated by this *InputLoader*.

23.4 LoadUniformSubSampleWithReplacement

This particular Class sub-samples with replacement the given dataset by using a uniform distribution. In the end a list of objects of Class *TabularDataSet* is created where each object has in its member dataset the new sub-sampled dataset.

The Class *LoadUniformSubSampleWithReplacement* inherits from the Class *InputLoader*.

Initialiser:
__init__(obj_name, single_split_length, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **single_split_length** (*int*) - This is the length of each new sub-sampled dataset: it is the number of samples that are contained in each *TabularDataSet* object.

Non-Parameters Members:

- **returns_dataset** (*bool*, True) - This is *True* since this input loader does generate a list of datasets.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

- **returns_env** (*bool*, False) - This is *False* since this input loader does not generate environments.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

Methods:

- **get_input(blocks, n_inputs_to_load, train_data=None, env=None)** - This method generates a list containing a number of *TabularDataSet* equal to **n_inputs_to_load**. Each dataset is sub-sampled with replacement using a uniform distribution from the given **train_data**.

  Parameters:

  - **blocks** (*list*) - This is a list containing the blocks for which we need to load the input. This is used only in some *InputLoader*. In this *InputLoader* it is not used.
  - **n_inputs_to_load** (*int*) - This is the number of datasets that will be sub-sampled.
  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns: This method returns two terms:

  - **splitted_datasets** (*list*) - This is a list containing objects of Class *TabularDataSet*.
  - *None* - This indicates that no environments were generated by this *InputLoader*.

23.5 LoadUniformSubSampleWithReplacementAndEnv

This particular Class sub-samples with replacement the given dataset by using a uniform distribution. In the end a list of objects of Class *TabularDataSet* is created where each object has in its member dataset the new sub-sampled dataset.

Moreover also a list of environments is returned so that this input loader can be used with metrics that use the environment, such as the *DiscountedReward* (to learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*).

The Class *LoadUniformSubSampleWithReplacementAndEnv* inherits from the Class *InputLoader*.

Initialiser:
__init__(obj_name, single_split_length, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **single_split_length** (*int*) - This is the length of each new sub-sampled dataset: it is the number of samples that are contained in each *TabularDataSet* object.

Non-Parameters Members:

- **returns_dataset** (*bool*, True) - This is *True* since this input loader does generate a list of datasets.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

- **returns_env** (*bool*, True) - This is *True* since this input loader does generate a list of environments.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

Methods:

- **get_input(blocks, n_inputs_to_load, train_data=None, env=None)** - This method generates two lists:
  - A list containing a number of *TabularDataSet* equal to **n_inputs_to_load**. Each dataset is sub-sampled with replacement using a uniform distribution from the given **train_data**.
  - A list containing a number of environments equal to **n_inputs_to_load**. Each environment is a deep copy of **env**.

  Note that this method simply creates two objects: one of Class *LoadUniformSubSampleWithReplacement* and one of Class *LoadSameEnv*.

  Then it proceeds just to call the method **get_input()** of these two objects.

  Parameters:

  - **blocks** (*list*) - This is a list containing the blocks for which we need to load the input. This is used only in some *InputLoader*. In this *InputLoader* it is not used.
  - **n_inputs_to_load** (*int*) - This is the number of datasets that will be sub-sampled, which also equals the number of deep copied environments.
  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.

- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns: This method returns two terms:

- **splitted_datasets** (*list*) - This is a list containing objects of Class *TabularDataSet*.
- **copied_envs** (*list*) - This is a list containing deep copies of the original environment **env**.

23.6 LoadDifferentSizeForEachBlock

This particular Class sub-samples with replacement the given dataset by using a uniform distribution, but it extracts for each block a different number of samples. In the end a list of objects of Class *TabularDataSet* is created where each object has in its member dataset the new sub-sampled dataset.

The Class *LoadDifferentSizeForEachBlock* inherits from the Class *InputLoader*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **returns_dataset** (*bool*, True) - This is *True* since this input loader does generate a list of datasets.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

- **returns_env** (*bool*, False) - This is *False* since this input loader does not generate a list of environments.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

Methods:

- **get_input(blocks, n_inputs_to_load, train_data=None, env=None)** - This method generates a list containing a number of *TabularDataSet* equal to **n_inputs_to_load**. Each dataset is sub-sampled with replacement using a uniform distribution from the given **train_data**.

  Note that in this case each dataset is made up of a different number of samples: this value is selected according to the value of **n_train_samples** present in each block. This parameter is an object of Class *HyperParameter* and thus can be mutated (i.e: it can be optimised by a *Tuner*).

  Note that the parameter **n_train_samples** is only present in offline model generation blocks.

  Parameters:

  - **blocks** (*list*) - This is a list containing the blocks for which we need to load the input. This is used only in some *InputLoader*. In this *InputLoader* it is used.
  - **n_inputs_to_load** (*int*) - This is the number of datasets that will be sub-sampled.
  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns: This method returns two terms:

  - **splitted_datasets** (*list*) - This is a list containing objects of Class *TabularDataSet*.
  - *None* - This indicates that no environments were generated by this *InputLoader*.

23.7 LoadDifferentSizeForEachBlockAndEnv

This particular Class sub-samples with replacement the given dataset by using a uniform distribution, but it extracts for each block a different number of samples. In the end a list of objects of Class *TabularDataSet* is created where each object has in its member dataset the new sub-sampled dataset.

Moreover also a list of environments is returned so that this input loader can be used with metrics that use the environment, such as the *DiscountedReward* (to learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*).

The Class *LoadDifferentSizeForEachBlockAndEnv* inherits from the Class *InputLoader*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **returns_dataset** (*bool*, True) - This is *True* since this input loader does generate a list of datasets.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

- **returns_env** (*bool*, True) - This is *True* since this input loader does generate a list of environments.

  To learn more about this check the documentation about the **Non-Parameters Members** of the Class *InputLoader*.

Methods:

- **get_input(blocks, n_inputs_to_load, train_data=None, env=None)** - This method generates two lists:
  - A list containing a number of *TabularDataSet* equal to **n_inputs_to_load**. Each dataset is sub-sampled with replacement using a uniform distribution from the given **train_data**.

    Note that in this case each dataset is made up of a different number of samples: this value is selected according to the value of **n_train_samples** present in each block. This parameter is an object of Class *HyperParameter* and thus can be mutated (i.e: it can be optimised by a *Tuner*).

    Note that the parameter **n_train_samples** is only present in offline model generation blocks.
  - A list containing a number of environments equal to **n_inputs_to_load**. Each environment is a deep copy of **env**.

  Note that this method simply creates two objects: one of Class *LoadDifferentSizeForEachBlock* and one of Class *LoadSameEnv*.

  Then it proceeds just to call the method **get_input()** of these two objects.

  Parameters:

  - **blocks** (*list*) - This is a list containing the blocks for which we need to load the input. This is used only in some *InputLoader*. In this *InputLoader* it is used.
  - **n_inputs_to_load** (*int*) - This is the number of datasets that will be sub-sampled, which also equals the number of deep copied environments.

- **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns: This method returns two terms:

- **splitted_datasets** (*list*) - This is a list containing objects of Class *TabularDataSet*.
- **copied_envs** (*list*) - This is a list containing deep copies of the original environment **env**.

---

24 metric.py

In the module *metric.py* the Classes *Metric*, *TDError*, *DiscountedReward*, *TimeSeriesRollingAverageDiscountedReward* and **SomeSpecificMetric** are implemented.

24.1 Metric

This Class is used as generic base class for all metrics. These are used to evaluate the good-ness of what was learnt in any block of a pipeline. These are also used in the *Tuner* for guiding the tuning procedure.

The *Metric* Classes check whether what they get as input is of the correct type: for example the *TDError* metric should check that **train_data** is an object of Class *TabularDataSet*.

The Class *Metric* is an *Abstract Class*, and so it inherits from *ABC*, but it also inherits from the Class *AbstractUnit*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **requires_dataset** (*bool*, None) - This is *True* if the metric requires a dataset to work.

  This is needed for checking the consistency of the consistency of the *Metric* with an *InputLoader*: this check is done in the *Tuner* via the method **is_metric_consistent_with_input_loader()**.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

- **requires_env** (*bool*, None) - This is *True* if the metric requires an environment to work.

  This is needed for checking the consistency of the consistency of the *Metric* with an *InputLoader*: this check is done in the *Tuner* via the method **_is_metric_consistent_with_input_loader()**.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

24.2 TDError

This Class implements a specific metric: given a dataset on which the model generation algorithm was learnt on it computes the TD error.

For each episode we compute the TD error, then from the TD error we compute its square. In the end we average over all the episodes.

Here smaller is better.

Note that the TDError can only be used for policies that have **regressor_type** different from *'generic_regressor'*, indeed an approximator of the Q-function is needed to compute the TDError.

The Class *TDError* inherits from the Class *Metric*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Non-Parameters Members:

- **requires_dataset** (*bool*, True) - This is *True* since in this metric we need a dataset to compute the TD error.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

- **requires_env** (*bool*, False) - This is *False* since in this metric we do not need an environment.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

- **eval_mean** (*float*, None) - This is the mean of the evaluation across the episodes.

- **eval_var** (*float*, None) - This is the variance of the evaluation across the episodes.

Methods:

- **evaluate(block_res, block=None, train_data=None, env=None)** - This method computes the TD error of the **block_res** over the provided **train_data**.

  This method computes for each episode the squared TD error and then returns the mean of the evaluations.

  Note that in this case the parameter **n_jobs** has no effect: the computation of the metric is vectorised.

  Parameters:

  - **block_res** (*BlockOutput*) - This must be an object of a Class inheriting from the Class *BlockOutput*.
  - **block** (*Block*, None) - This must be an object of a Class inheriting from the Class *Block*.
  - **train_data** (*TabularDataSet*, None) - This must be an object of a Class inheriting from the Class *TabularDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **ag_eval** (*float*) - This is the mean TD Error across the provided episodes.

- **which_one_is_better(block_1_eval, block_2_eval)** - This method is needed to decide which of the two blocks is best. Since this metric (*TDError*) is better if minimised then the best block is the one with the lowest evaluation.

  This method compares the value of the member **block_eval**, which are the two values provided as parameters.

  This is needed in the Class *Tuner* and in the automatic blocks (e.g: *AutoModelGeneration*) to decide among best tuned blocks, without having to distinguish between a maximisation problem or a minimisation problem.

  Parameters:

  - **block_1_eval** (*float*) - This is the value of the member **block_eval** of an object of a Class inheriting from the Class *Block*.
  - **block_2_eval** (*float*) - This is the value of the member **block_eval** of an object of a Class inheriting from the Class *Block*.

  Returns:

  - (*int*) - This is *0* if the first block is the best one (i.e: it has the lowest evaluation), else it is *1*.

24.3 DiscountedReward

This Class implements a specific metric: given an environment and a policy (i.e: the output of a model generation block) it computes the average discounted reward.

For each episode we compute the discounted reward and then we average over the episodes.

Here bigger is better.

The Class *DiscountedReward* inherits from the Class *Metric*.

Initialiser:
__init__(obj_name, n_episodes, env_dict_of_params=None, batch=False, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **n_episodes** (*int*) - This must be a positive a positive integer and it is the number of episodes for which to evaluate the agent.
- **env_dict_of_params** (*dict*, None) - This is a dictionary containing some members of the environment that you may want to modify only for the evaluation phase. The key must be the name of the member of the environment and the value must be the new value to be assigned to such member.

  For example if you have an LQG environment you may want to include a noise on the controller only for testing, to simulate real world situations.

  To understand how this work look at the documentation of the methods **set_params()** and **get_params()** of the Class *BaseEnvironment*.
- **batch** (*bool*, False) - This is a boolean:
  - If *True* then the discounted reward is computed in batch mode, namely we copy the environment for a number of times equal to **n_episodes**, and we make one step of advance at the time for all episodes.

    If **n_jobs** is greater than 1 then also the call to the method **step()** of the environment **env** is performed in parallel. Note that this may slow things down if a single call to the method **step()** is not enough computationally expensive.

    Remark: The batch mode cannot be applied on all model generation blocks: it can be applied only to those model generation blocks whose policy has a non *None* value for the member **approximator**.

    Indeed the method **draw_action()** of a MushroomRL policy can only be called on a single state at the time. Instead the method **predict()** of the **approximator** can be called on as many states as we like.

    This happens for all Classes inheriting from the Class *ModelGenerationMushroomOffline* and for the Classes inheriting from the Class *ModelGenerationMushroomOffline* that have **deterministic_output_policy** equal to *True*.
  - If *False* then the discounted reward is computed either in serial or in parallel: if serial then we go through one single episode at the time, if parallel we divide the episodes over the selected number of jobs.

Non-Parameters Members:

- **requires_dataset** (*bool*, False) - This is *False* since in this metric we do need a dataset to compute the discounted reward.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

- **requires_env** (*bool*, True) - This is *True* since in this metric we need an environment to compute the discounted reward.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

- **eval_mean** (*float*, None) - This is the mean of the evaluation across the episodes.

- **eval_var** (*float*, None) - This is the variance of the evaluation across the episodes.

Methods:

- **_evaluate_some_episodes(block_res, local_n_episodes, env=None)** - This method computes the average discounted reward for a number of episodes equal to **local_n_episodes**.

  Parameters:

  - **block_res** (*BlockOutput*) - This must be an object of a Class inheriting from the Class *BlockOutput*.
  - **local_n_episodes** (*int*) - This is the number of episodes for which we need to evaluate **block_res**.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **total_rew** (*list*) - This is a list containing a number of elements equal to **local_n_episodes**. Each element is the average discounted reward over one episode.

- **_non_batch_eval(block_res, train_data=None, env=None)** - We create a number of environments equal to the **n_jobs** (by deep copy of the original environment). Then the method **_evaluate_some_episodes()** is called in parallel: in particular each call is associated with one of the deep copied environments and each call has to do a number of evaluations equal for a certain number of episodes, namely:

  - From the first to the second last call of this method we have to evaluate for a number of episodes equal to the integer part of:
  $$\frac{\textbf{n\_episodes}}{\textbf{n\_jobs}}$$
  - For the last call of this method we have to evaluate for a number of episodes equal to:
  $$1 - \left(\textbf{(n\_jobs -1)} \left\lfloor \frac{\textbf{n\_episodes}}{\textbf{n\_jobs}} \right\rfloor\right)$$

  Parameters:

  - **block_res** (*BlockOutput*) - This must be an object of a Class inheriting from the Class *BlockOutput*.
  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

- **unlisted_evals** (*list*) - This is a list containing **n_episodes** elements: each element is the average discounted reward over one episode.

- **_batch_eval_parallel_step(block_res, train_data=None, env=None)** - This method computes the discounted reward for **n_episodes**. A number of episodes equal to **n_episodes** is created (by deep copy of the original environment) and then trajectories are rolled out in batch (i.e: for all environments at the same time, one step at the time).

  Moreover this method calls in parallel the method **step()** of each of the **n_episodes**. This may slow things down if a single call to the method **step()** is not enough computationally expensive.

  For more information on the batch evaluation see the **Remark** about the **batch** parameter in the **Parameters** section of the Class *DiscountedReward*.

  Parameters:

  - **block_res** (*BlockOutput*) - This must be an object of a Class inheriting from the Class *BlockOutput*.
  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **tmp_rew** (*numpy.array*) - This is an array that has **n_episodes** components: in each component the average discounted reward over an episode is contained.

- **_batch_eval_no_parallel(block_res, train_data=None, env=None)** - This method computes the discounted reward for **n_episodes**. A number of episodes equal to **n_episodes** is created (by deep copy of the original environment) and then trajectories are rolled out in batch (i.e: for all environments at the same time, one step at the time).

  Parameters:

  - **block_res** (*BlockOutput*) - This must be an object of a Class inheriting from the Class *BlockOutput*.
  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **tmp_rew** (*numpy.array*) - This is an array that has **n_episodes** components: in each component the average discounted reward over an episode is contained.

- **evaluate(block_res, block=None, train_data=None, env=None)** - This method computes the discounted reward of the **block_res** using the provided **env**.

  First the **env** parameters are modified according to the provided **env_dict_of_params**. Then this method computes for each episode the discounted reward and then returns the mean of the evaluations.

  This method computes the evaluations calling 3 other methods, namely:

  - If **batch** is *True* and **n_jobs** is 1 then the method **_batch_eval_no_parallel()** is called.
  - If **batch** is *True* and **n_jobs** is greater than 1 then the method **_batch_eval_parallel_step()** is called.
  - If **batch** is *False* then the method **_non_batch_eval()** is called.

Note that **n_jobs** cannot be higher than **n_episodes**: if higher it will be lowered to the same value of **n_episodes**.

Parameters:

- **block_res** (*BlockOutput*) - This must be an object of a Class inheriting from the Class *BlockOutput*.
- **block** (*Block*, None) - This must be an object of a Class inheriting from the Class *Block*.
- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- **ag_eval** (*float*) - This is the mean discounted reward across the provided episodes.

• **which_one_is_better(block_1_eval, block_2_eval)** - This method is needed to decide which of the two blocks is best. Since this metric (*DiscountedReward*) is better if maximised then the best block is the one with the highest evaluation.

This method compares the value of the member **block_eval**, which are the two values provided as parameters.

This is needed in the Class *Tuner* and in the automatic blocks (e.g: *AutoModelGeneration*) to decide among best tuned blocks, without having to distinguish between a maximisation problem or a minimisation problem.

Parameters:

- **block_1_eval** (*float*) - This is the value of the member **block_eval** of an object of a Class inheriting from the Class *Block*.
- **block_2_eval** (*float*) - This is the value of the member **block_eval** of an object of a Class inheriting from the Class *Block*.

Returns:

- (*int*) - This is *0* if the first block is the best one (i.e: it has the highest evaluation), else it is *1*.

24.4 TimeSeriesRollingAverageDiscountedReward

This Class implements a specific metric: given an environment and a policy (i.e: the output of a model generation block) it computes the rolling average discounted reward. Suppose we have an environment in which each horizon is made up of one day, then:

- We learn the block on the first N days and evaluate it from the $(N+1)$-th day to the $(N+M)$-th day.
- We learn the block on the first $N+M$ days and evaluate it from the $(N+M)$-th day to the $(N+2M)$-th day.
- We learn the block on the first $N+2M$ days and evaluate it from the $(N+2M)$-th day to the (N+3M)-th day.
- And so on and so forth...

The final result is the average of the result of the above steps.

In order to be able to use this metric the provided environment must have an attribute for selecting the time step at which to start the episode.

The environment must have two members:

- **min_time_step_for_time_series_evaluation**
- **max_time_step_for_time_series_evaluation**

These two members must limit the values the time step can take when calling the method **reset()** of the environment.

Here bigger is better.

The Class *TimeSeriesRollingAverageDiscountedReward* inherits from the Class *Metric*.

Initialiser:
__init__(obj_name, n_episodes_train, n_evaluations, n_episodes_eval, n_episodes_per_fit=None, data_gen_block=None, env_dict_of_params=None, batch=False, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

Parameters:

- **n_episodes_train** (*int*) - This is a number and it is the starting number of episodes to use. We train the block on **n_episodes** and we evaluate it on **n_episodes_eval**, then we train the block on (**n_episodes_train** + **n_episodes_eval**) and we evaluate it on **n_episodes_eval**, and so on and so forth.
- **n_evaluations** (*int*) - This is a number and it is the total number of evaluations to perform: it is the number of rolling windows.
- **n_episodes_eval** (*int*) - This is a number and it is the number of episodes for evaluation to use in each rolling window.
- **n_episodes_per_fit** (*int*, None) - This is a parameter needed for learning the online *ModelGeneration* block, and it must be less than **n_episodes**. It is not used if the block to be evaluated is an offline model generation block.
- **data_gen_block** (*DataGeneration*, None) - This must be an object of a Class inheriting from the Class *DataGeneration*. This is used if the block to evaluate is an offline *ModelGeneration* block in which case we need a way to extract data from the given environment.

- **env_dict_of_params** (*dict*, None) - This is a dictionary containing some members of the environment that you may want to modify only for the evaluation phase. The key must be the name of the member of the environment and the value must be the new value to be assigned to such member.

  For example if you have an LQG environment you may want to include a noise on the controller only for testing, to simulate real world situations.

  To understand how this work look at the documentation of the methods **set_params()** and **get_params()** of the Class *BaseEnvironment*.

Non-Parameters Members:

- **requires_dataset** (*bool*, False) - This is *False* since in this metric we do need a dataset to compute the rolling average discounted reward.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

- **requires_env** (*bool*, True) - This is *True* since in this metric we need an environment to compute the rolling average discounted reward.

  To learn more about this check the documentation about the method **is_metric_consistent_with_input_loader()** of the Class *Tuner*.

- **eval_mean** (*float*, None) - This is the mean of the evaluation across the rolling windows.

- **eval_var** (*float*, None) - This is the variance of the evaluation across the rolling windows.

Methods:

- **_online_blocks_time_series_rolling_eval(block, rolling_window_idx, original_lower_bound, train_data=None, env=None)**
  - This method only works on online *ModelGeneration* blocks and as such I might be calling methods that are only present in those blocks.

  This method computes the discounted reward over a single rolling window:

  - Since this method works on online *ModelGeneration* blocks we directly learn the **block** over the provided **env**.
  - Finally the environment is used to compute the average discounted reward of the learnt **block**. The evaluation is performed on unseen data.

  To make sure we use different data between training and testing we set appropriately in each call to this method the value of the parameters **min_time_step_for_time_series_evaluation** and **max_time_step_for_time_series_evaluation** of the **env**.

  Parameters:

  - **block** (*Block*, None) - This must be an object of a Class inheriting from the Class *Block*.
  - **rolling_window_idx** (*int*) - This is an integer and it represent the current rolling window. It is used to set **min_time_step_for_time_series_evaluation** and **max_time_step_for_time_series_evaluation**.

- **original_lower_bound** (*int*) - This is an integer and it represent the original value of **min_time_step_for_time_series_evaluation**, that is: the value of such member at the start of the call of the method **evaluate()**.

  This may be used by the user to restrict the evaluation only to some time steps. For example one may want to only consider time steps that come after the 100th time step, in which case this parameter would assume the value of 101.
- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- (*float*) - This is the mean discounted reward across the provided episodes of a single rolling window.
- (*float*) - This is the variance of the discounted reward across the provided episodes of a single rolling window.

• **_offline_blocks_time_series_rolling_eval(block, rolling_window_idx, original_lower_bound, train_data=None, env=None)**
- This method only works on offline *ModelGeneration* blocks and as such I might be calling methods that are only present in those blocks.

This method computes the discounted reward over a single rolling window:

- Since this method works on offline *ModelGeneration* blocks then we need to extract a dataset from the environment. This is done via the parameter **data_gen_block** of the **TimeSeriesRollingAverageDiscountedReward** object.
- After the dataset is extracted the **block** is learnt over the extracted dataset.
- Finally the environment is used to compute the average discounted reward of the learnt **block**. The evaluation is performed on unseen data.

To make sure we use different data between training and testing we set appropriately in each call to this method the value of the parameters **min_time_step_for_time_series_evaluation** and **max_time_step_for_time_series_evaluation** of the **env**.

Parameters:

- **block** (*Block*, None) - This must be an object of a Class inheriting from the Class *Block*.
- **rolling_window_idx** (*int*) - This is an integer and it represent the current rolling window. It is used to set **min_time_step_for_time_series_evaluation** and **max_time_step_for_time_series_evaluation**.
- **original_lower_bound** (*int*) - This is an integer and it represent the original value of **min_time_step_for_time_series_evaluation**, that is: the value of such member at the start of the call of the method **evaluate()**.

  This may be used by the user to restrict the evaluation only to some time steps. For example one may want to only consider time steps that come after the 100th time step, in which case this parameter would assume the value of 101.
- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- (*float*) - This is the mean discounted reward across the provided episodes of a single rolling window.

- (*float*) - This is the variance of the discounted reward across the provided episodes of a single rolling window.

- **evaluate(block_res, block=None, train_data=None, env=None)** - This method computes the rolling average discounted reward of the **block** using the provided **env**.

  Here **block_res** is not used since for each rolling window the **block** needs to be learnt from scratch.

  This method calls either the method **_online_blocks_time_series_rolling_eval()** or the method **_offline_blocks_time_series_rolling_eval()**, depending on the **pipeline_type** of the **block**.

  For each rolling window a different environment is created: this is done by deep copying the original **env**.

  If **n_jobs** is greater than 1 this method calls one of the aforementioned methods in parallel: indeed each rolling window can be evaluated in parallel.

  Inside each of the two aforementioned methods an object of Class *DiscountedReward* is created: the call to this inner metric is not done in parallel to avoid a parallel loop inside a parallel loop, as it could cause some issues. For example a process fork of a process fork is not possible.

  Parameters:

  - **block_res** (*BlockOutput*) - This must be an object of a Class inheriting from the Class *BlockOutput*.
  - **block** (*Block*, None) - This must be an object of a Class inheriting from the Class *Block*.
  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **ag_eval** (*float*) - This is the mean rolling average discounted reward across the rolling windows.

- **which_one_is_better(block_1_eval, block_2_eval)** - This method is needed to decide which of the two blocks is best. Since this metric (*TimeSeriesRollingAverageDiscountedReward*) is better if maximised then the best block is the one with the highest evaluation.

  This method compares the value of the member **block_eval**, which are the two values provided as parameters.

  This is needed in the Class *Tuner* and in the automatic blocks (e.g: *AutoModelGeneration*) to decide among best tuned blocks, without having to distinguish between a maximisation problem or a minimisation problem.

  Parameters:

  - **block_1_eval** (*float*) - This is the value of the member **block_eval** of an object of a Class inheriting from the Class *Block*.
  - **block_2_eval** (*float*) - This is the value of the member **block_eval** of an object of a Class inheriting from the Class *Block*.

  Returns:

  - (*int*) - This is *0* if the first block is the best one (i.e: it has the highest evaluation), else it is *1*.

24.5 SomeSpecificMetric

This Class implements a specific metric: this is a placeholder to use in blocks for which there are no metrics, or for which you do not need a metric.

The Class *SomeSpecificMetric* inherits from the Class *Metric*.

Initialiser:
__init__(obj_name, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process')

---

25 tuner.py

In the module *tuner.py* the Class *Tuner* is implemented.

25.1 Tuner

This Class is used as base for all block tuning: the specific tuning algorithms inherit from this Class. Any block that wants to be tuned needs to use a tuner: starting from a generic configuration of hyper-parameters *Tuner* objects find a better hyper-parameters configuration that provides a better metric evaluation.

Tuner objects can be applied to blocks that inherit from the Classes:

- *ModelGeneration*
- *RLPipeline*

Tuner objects can only be applied to blocks inheriting from the Classes mentioned above, but there is one more restriction: such blocks must have the parameter **is_parametrised** equal to *True*.

The Class *Tuner* is an *Abstract Class*, and so it inherits from *ABC*, but it also inherits from the Class *AbstractUnit*.

Initialiser:
__init__(block_to_opt, eval_metric, input_loader, obj_name, create_explanatory_heatmap=False, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', output_save_periodicity=25)

Parameters:

- **block_to_opt** (*Block*) - This must be an object inheriting from the Class *Block*. In particular it can only be a *FeatureEngineering* block, or a *ModelGeneration* block, or a *RLPipeline* block.

  This block moreover must have the parameter **is_parametrised** equal to *True*.

  This is the block whose hyper-parameters are going to be tuned.

- **eval_metric** (*Metric*) - This must be an object inheriting from the Class *Metric*. This is the metric that is going to be used to rank the different tuned blocks in the tuner.

  The user must make sure that the provided metric is a sensible choice for the provided block.

- **input_loader** (*InputLoader*) - This must be an object inheriting from the Class *InputLoader*. This object is going to generate the right **train_data** and-or **env** for each block, in the tuning procedure.

  The user must make sure that the provided input loader is a sensible choice for the provided block.

- **output_save_periodicity** (*int*, 25) - This is a positive integer and it represents the frequency with which to save the blocks as the tuning procedure takes place.

  Note that if the member **checkpoint_log_path** is not set then nothing will be saved.

  This is fundamental in order to create the explanatory heatmap of the hyper-parameters, which is generated by the method **create_explanatory_heatmap_hyperparameters()** at the end of the tuning procedure. If too few blocks are saved throughout the tuning procedure, then the same few blocks will go in input to the method **create_explanatory_heatmap_hyperparameters()** and so the resulting heatmap quality might be low.

- **create_explanatory_heatmap** (*bool*, False) - This is a boolean and if *True* then at the end of the call of the method **tune()** the method **create_explanatory_heatmap_hyperparameters()** is going to be called: this creates an explanatory heatmap of the hyper-parameters.

  To obtain the best possible result with the heatmap you should only create when **output_save_periodicity** is equal to 1.

Non-Parameters Members:

- **is_tune_successful** (*bool*, False) - This is used to know whether or not the tuner finished with no errors. If everything went smooth then it will be set to *True* before the end of the method **tune()**.

Methods:

- **is_metric_consistent_with_input_loader()** - This method returns *True* if the **eval_metric** and the **input_loader** are consistent.

  These two are consistent when the metric requires in input something that the input loader will return. More precisely:

  - If **eval_metric** has **requires_dataset** equal to *True* then **input_loader** must have **returns_dataset** equal to *True*.
  - If **eval_metric** has **requires_env** equal to *True* then **input_loader** must have **returns_env** equal to *True*.

  If this method returns *False* then the tuning procedure cannot take place.

  Returns:

  - (*bool*) - This is *True* if the **eval_metric** and the **input_loader** are consistent, else it is *False*.

- **tune(train_data=None, env=None)** - This method first checks that the **eval_metric** and the **input_loader** are consistent, by calling the method **is_metric_consistent_with_input_loader()**.

  Then it checks that the **block_to_opt** has the parameter **is_parametrised** equal to *True*. Note that one can set **is_parametrised** equal to *False* in two cases:

  - When one doesn't want to tune the hyper-parameters of such block.
  - When such block cannot be tuned since it has no hyper-parameters.

  If **block_to_opt** has the parameter **is_parametrised** equal to *False*: The **block_to_opt** is learnt over the entire provided **train_data** and **env**. Then:

  - If the **block_to_opt** was learnt successfully: this agent, and its evaluation, are returned by this method.
  - If the **block_to_opt** was **not** learnt successfully: this method returns value *None* both for the agent and for its evaluation.

  If **block_to_opt** has the parameter **is_parametrised** equal to *True*: nothing is returned, hence the computation will proceed in a specific *Tuner* object which will inherit from the Class *Tuner*.

  Note that in this case a folder for saving the tuned agents, and their result, is created: the folder name is given by the concatenation of the **obj_name** of the *Tuner* object followed by the current time and date.

  If a folder with such a same name already exists the computation stops and does not proceed any further.

  Parameters:

- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- If **is_parametrised** is equal to *True*: nothing is returned.
- If **is_parametrised** is equal to *False*:
  * **best_agent** (*Block*), **best_agent_eval** (*float*) - This is the return value in case the call to the method **learn()** of the **block_to_opt** succeeds.
  * (*None*, *None*) - This is the return value in case the call to the method **learn()** of the **block_to_opt** fails.

• **update_verbosity(new_verbosity)** - This method calls the *base* method **update_verbosity()** implemented in the Class *AbstractUnit*. Then it calls the method **update_verbosity()** for **block_to_opt** and **input_loader**.

Parameters:

- **new_verbosity** (*int*) - This must be a positive integer representing the new **verbosity**.

• **create_explanatory_heatmap_hyperparameters()** - This method is called at the end of the call of the method **tune()** only when **create_explanatory_heatmap** is set to *True*.

Starting from the result of the tuning procedure constructs a heatmap in which we can see the evaluation of an agent, when changing the value of two of its hyper-parameters, while keeping the other hyper-parameters value fixed to the value of the optimal agent obtained in the tuning procedure.

The agent evaluation seen on the heatmap is the predicted value for the new hyper-parameters configuration measured according to the **eval_metric** provided in the tuner.

Specifically this method:

- Loads the data saved by the method **tune()**.
- Loads the best agent obtained by the tuning procedure.
- From the loaded agents it extracts a dataset where the features are the hyper-parameters configurations and the target is the obtained block evaluation.
- It fits a *CatBoostRegressor* on such dataset.
- For each pair of hyper-parameters that were tuned a $2-D$ grid is constructed and it is used to generate a new dataset. A new dataset is generated for each pair of hyper-parameters that were tuned.

  Each dataset has fixed values for the hyper-parameters that were not tuned, equal to the values of the optimal found hyper-parameters overall, whereas for the hyper-parameters that are in the current pair the values change over a grid of points.

  Each dataset is passed to the method **predict()** of the previously fitted *CatBoostRegressor*.
- A heatmap is created by plotting such prediction. To do this the **plotly** library is used: an *html* file is generated containing the interactive heatmap where the effect of changing the value of the hyper-parameters pairs can be observed.

  Part of the code that creates the heatmap is a re-adaptation of the following code:
  cf. https://plotly.com/python/custom-buttons/

At the end an *html* file, containing the heatmap, is saved.

26 tuner_optuna.py

In the module *tuner_optuna.py* the Class *TunerOptuna* is implemented.

26.1 TunerOptuna

This Class implements an hyper-parameter optimisation algorithm, namely it provides access to the *Optuna* library.
cf. https://optuna.readthedocs.io/en/stable/index.html
cf. https://arxiv.org/abs/1907.10902

The Class *TunerOptuna* inherits from the Class *Tuner*.

Initialiser:
__init__(block_to_opt, eval_metric, input_loader, obj_name, create_explanatory _heatmap=False sampler='TPE', n_trials=100, max_time_seconds=3600, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_jobs=1, job_type='process', output_save_periodicity=25)

Parameters:

- **sampler** (*str*, 'TPE') - This is a string representing the sampler to use out of the ones provided by *Optuna*. It can be:
  - *'CMA-ES'*, in which case th covariance matrix adaptation evolution strategy is used.
  - *'TPE'*, in which case the tree parzen estimator is used.
  - *'RANDOM'*, in which case random search is used.
  - *'GRID'*, in which case grid search is used.

  Note that some samplers have some requirements: for example the *'CMA-ES'* sampler does not work on categorical parameters.
  cf. https://optuna.readthedocs.io/en/stable/tutorial/10_key_features/003_efficient_optimization_algorithms.html#sampling-algorithms
- **n_trials** (*int*, 100) - This must be a positive integer representing the number of trials to do.
- **max_time_seconds** (*int*, 3600) - This must be a positive integer representing the maximum allowed time in seconds.

Non-Parameters Members:

- **optuna_object_sampler** (*optuna.samplers.sampler._base.BaseSampler*) - This is a sampler object from the *Optuna* library. It is an object constructed by using the parameter **sampler**.
- **opt_direction** (*str*) - This is a string and it is either *'maximize'* or *'minimize'*.

  It is *'maximize'* if the metric of the block needs to be maximised, else it is *'minimize'*.

Methods:

- **_objective(trial)** - This method is called by *Optuna* throughout the study. This method takes the current agent and it updates its hyper-parameters according to the **trial** suggested values.

Then the new agent is learnt over the current **train_data** and **env** and it is evaluated according to the **eval_metric**.

Note that if the block to be evaluate is an object of a Class inheriting from the Class *RLPipeline* and such pipeline ends with an object of a Class inheriting from the Class *ModelGeneration* then the **train_data** and the **env** used in the evaluation are those situated in the object obtained in output from the method **learn()** of the pipeline. Why is this done? Because if we performed feature engineering then the learnt policy cannot be evaluated in the old **train_data** and **env** as the observation space and the action space are different, and so we need the latest version of the **train_data** and **env**.

After each number of trials equal to **output_save_periodicity** we save the agent and its result in a *pickle* file.

Moreover also every new best agent, and its result, are saved.

Finally at the end of this method the new **train_data** and **env**, obtained from the **input_loader**, are constructed.

If everything went smoothly the new agent evaluation is returned, else *None* is returned.

Note that here, unless **verbosity** is greater than or equal to 4 we silence the agents: otherwise if we are in debug mode we print everything.

Parameters:

- **trial** (*optuna.trial.Trial*) - This object provides interfaces to get suggestions for the new values of the hyper-parameters.
  cf. https://optuna.readthedocs.io/en/stable/reference/generated/optuna.trial.Trial.html

  This is used by *Optuna*.

Returns:

- If the new agent was **not** learnt properly: *None* is returned.
- If the new agent was learnt properly the return value is: **tmp_agent_eval** (*float*) -This is the evaluation, according to **eval_metric**, of the new agent.

• **tune(train_data=None, env=None)** - This method calls the *base* method **tune()** implemented in the Class *Tuner*. Then we check the result of such call:

  - If what we get is *None*, it means that such method did not return anything, and so we are free to proceed.
  - Else we return immediately the results we got passed down from such method.

  First the **input_loader** generates the data for the first agent and then an *Optuna* study is created: this is an object of Class *optuna.study.Study* and it coordinates the tuning procedure.

  Now the method **optimize()** of such study is called: this takes care of the whole hyper-parameter optimisation procedure.

  After this we can extract the best trial, according to the **eval_metric**, and create the final agent with the best set of hyper-parameters that was found. This new agent is also saved to a *pickle* file.

  Now if **create_explanatory_heatmap** is equal to *True*, the heatmap explaining the impact of the hyper-parameters on the evaluation of **block_to_opt** is created by calling the method **create_explanatory_heatmap_hyperparameters(**

  Finally the best agent and its evaluation are returned.

Parameters:

- **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
- **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

Returns:

- If everything went smoothly we return: **best_agent** (*Block*), **best_agent_eval** (*float*) - These are the best agent and its evaluation.
- If there was an error we return: *None*, *None*.

---

27 tuner_genetic.py

In the module *tuner_genetic.py* the Class *TunerGenetic* is implemented.

27.1 TunerGenetic

This Class implements a population based tuner, namely it implements a Genetic Algorithm.

The Class *TunerGenetic* inherits from the Class *Tuner*.

Initialiser:
__init__(block_to_opt, eval_metric, input_loader, obj_name, create_explanatory_heatmap=False, seeder=2, log_mode='console', checkpoint_log_path=None, verbosity=3, n_agents=10, n_generations=100, prob_point_mutation=0.5, tuning_mode='best_performant_elitism', pool_size=None, n_jobs=1, job_type='process', output_save_periodicity=25)

Parameters:

- **n_agents** (*int*, 10) - This must be a positive integer representing the number of agents in each generation.
- **n_generations** (*int*, 100) - This must be a positive integer representing the number of generations of the genetic algorithm.
- **prob_point_mutation** (*float*, 0.5) - This is the probability of mutating a certain hyper-parameter of **block_to_opt**.
- **tuning_mode** (*str*, 'best_performant_elitism') - This is a string and it represents the tuning mode. It can assume one of three values:
  - If *'no_elitism'* is selected then the best agent of each generation is not kept as is across generations but it undergoes mutation.
  - If *'best_performant_elitism'* is selected then the best agent so far is added to each generation both as is and also mutated.
  - If *'pool_elitism'* is selected then the most performant **pool_size** agents are kept across generations and are used to generate the new offspring by mutating them.

  In the first two cases we perform tournament selection (note that we always reserve a spot for the best agent of the previous generation (which will be mutated)).
- **pool_size** (*int*, None) - This is a positive integer and it represents the number of best agents to use in each generation as starting point for obtaining the next generation.

  This is only needed when **tuning_mode** is equal to *'pool_elitism'*.

  Note that **pool_size** must divided exactly **n_agents**: otherwise an exception is raised.

Non-Parameters Members:

- **trial_number** (*int*, 0) - This is an integer used for keeping track of how many trials (agents) are being done.

Methods:

- **_get_agent_data(current_agent, train_data=None, env=None)** - This method creates the appropriate **train_data** and-or **env** for the provided **current_agent** by using the **input_loader**.

  Parameters:

  - **current_agent** (*Block*) - This is the block for which we need to construct the **train_data** and-or **env** which will be used by the block for learning.
  - **train_data** (*BaseDataSet*, None) - This is the original **train_data** as passed in input to the method **tune()** of the *Tuner*.
  - **env** (*BaseEnvironment*, None) - This is the original **env** as passed in input to the method **tune()** of the *Tuner*.

  Returns:

  - **tmp_agent_train_data** (*BaseDataSet*) - This can be *None* (in case no dataset was needed to be extracted for the provided block **current_agent**).

    Otherwise if it is not *None* it will be an object of a Class inheriting from the Class *BaseDataSet*.
  - **tmp_agent_env** (*BaseEnvironment*) - This can be *None* (in case no environment was needed to be extracted for the provided block **current_agent**).

    Otherwise if it is not *None* it will be an object of a Class inheriting from the Class *BaseEnvironment*.

  Note that all the above values are equal to *None* if something went wrong in the execution of this method.

- **_mutate_gather_data_and_env(current_gen_n, current_gen_length, tmp_agent_to_tune, train_data=None, env=None, first_mutation=False)**
  -This method takes as input an agent, it deep copies it, then:

  - The agent hyper-parameters are mutate by calling the method **_mutate()**
  - The agent is seeded with a different seed by calling its method **set_local_prng()**.
  - The **train_data** and-or **env** needed to learn the newly mutated agent are extracted by calling the method **_get_agent_data()**

  Note that here, unless **verbosity** is greater than or equal to 4 we silence the agents: otherwise if we are in debug mode we print everything.

  Parameters:

  - **current_gen_n** (*int*) - This the number of the current generation. It is solely used for re-naming each newly mutated agent.
  - **current_gen_length** (*int*) - This is the current generation length: it is the number of agents in the current generation. This is used for re-naming each newly mutated agent and for setting a different seed for each agent.
  - **tmp_agent_to_tune** (*Block*) - This must be an object of a Class inheriting from the Class *Block*. We deep copy this agent, and mutate its hyper-parameters.
  - **train_data** (*BaseDataSet*, None) - This is the original **train_data** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **train_data** for the newly mutated agent.
  - **env** (*BaseEnvironment*, None) - This is the original **env** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **env** for the newly mutated agent.

- **first_mutation** (*bool*, False) - This is a boolean and it is used to decide how to mutate the hyper-parameters of the agent. This is passed to the method **_mutate()**.

  To see its effect see the documentation about the Class *Integer*, *Real*, *Categorical* in the section explaining about the parameter **type_of_mutation**.

Returns:

- **tmp_agent** (*Block*) - This is the newly mutated agent.
- **tmp_agent_train_data** (*BaseDataSet*) - This can be *None* (in case no dataset was needed to be extracted for the newly mutated agent).

  Otherwise if it is not *None* it will be an object of a Class inheriting from the Class *BaseDataSet*: this is used in the method **learn()** of the newly mutated agent.
- **tmp_agent_env** (*BaseEnvironment*) - This can be *None* (in case no environment was needed to be extracted for the newly mutated agent).

  Otherwise if it is not *None* it will be an object of a Class inheriting from the Class *BaseEnvironment*: this is used in the method **learn()** of the newly mutated agent.

Note that all the above values are equal to *None* if something went wrong in the execution of this method.

- **_learn_and_evaluate(tmp_agent, tmp_agent_train_data, tmp_agent_env)** - This method calls the method **learn()** of **tmp_agent** by passing to it **tmp_agent_train_data** and **tmp_agent_env**.

Then if the learning was not successful the **tmp_agent** is returned. Else if the learning was successful: the learn agent is evaluated by calling the method **_evaluate()**. This evaluation is added to the parameter **block_eval** of **tmp_agent**.

Note that if the block to be evaluate is an object of a Class inheriting from the Class *RLPipeline* and such pipeline ends with an object of a Class inheriting from the Class *ModelGeneration* then the **train_data** and the **env** used in the evaluation are those situated in the object obtained in output from the method **learn()** of the pipeline. Why is this done? Because if we performed feature engineering then the learnt policy cannot be evaluated in the old **train_data** and **env** as the observation space and the action space are different, and so we need the latest version of the **train_data** and **env**.

Now if **trial_number** divides **output_save_periodicity** the learnt agent and its result are saved as *pickle* files.

A check here is also made to update the best agent found so far: we compare the new **block_eval** with that of the best agent found so far.

Finally **tmp_agent** is returned.

Parameters:

- **tmp_agent** (*Block*) - This is the newly mutated agent which we need to learn.
- **tmp_agent_train_data** (*BaseDataSet*) - This can be *None* (in case no dataset was needed to be extracted for the newly mutated agent).

  Otherwise if it is not *None* it will be an object of a Class inheriting from the Class *BaseDataSet*: this is used in the method **learn()** of the newly mutated agent.

- **tmp_agent_env** (*BaseEnvironment*) - This can be *None* (in case no environment was needed to be extracted for the newly mutated agent).

  Otherwise if it is not *None* it will be an object of a Class inheriting from the Class *BaseEnvironment*: this is used in the method **learn()** of the newly mutated agent.

Returns:

- **tmp_agent** (*Block*) - This is the agent that was learnt: it is the same object as the one provided as parameter except that its parameter **block_eval** has been changed.

  This has value *None* if something went wrong in the execution of this method.

• **_no_elitism_or_best_performant_elitism_common(agents_population, preserve_best_agent, train_data=None, env=None)** - This method is used when **tuning_mode** is not *'pool_elitism'*.

This method performs the tuning from the second to the last generation: it takes as input the first generation and then it returns as output the last generation.

For each generation the new agents population is constructed by tournament selection by repeatedly calling the method **_select()**: we take subsets of 3 agents and pick the best among the three agents. We reserve one spot for the best performing agent of the previous generation.

Once the new generation is constructed:

- The **train_data** and-or **env** are created, by repeatedly calling the method **_mutate_gather_data_and_env()**.
- The agents are learnt and evaluated, by repeatedly calling the method **_learn_and_evaluate()**.

Parameters:

- **agents_population** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the first generation.
- **preserve_best_agent** (*bool*) - This is a boolean and it is *True* if **tuning_mode** is equal to *'best_performant_elitism'*, else it is *False*.
- **train_data** (*BaseDataSet*, None) - This is the original **train_data** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **train_data** for the generation.
- **env** (*BaseEnvironment*, None) - This is the original **env** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **env** for the generation.

Returns:

- **agents_population** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the last generation.

  This has value *None* if something went wrong in the execution of this method.

• **_no_elitism(agents_population, train_data=None, env=None)** - This method is used when **tuning_mode** is equal to *'no_elitism'*.

This method simply calls the method **_no_elitism_or_best_performant_elitism_common()** with **preserve_best_agent** equal to *False*.

Parameters:

– **agents_population** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the first generation.
– **train_data** (*BaseDataSet*, None) - This is the original **train_data** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **train_data** for the generation.
– **env** (*BaseEnvironment*, None) - This is the original **env** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **env** for the generation.

Returns:

– **agents_population** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the last generation.

  This has value *None* if something went wrong in the execution of this method.

- **_best_performant_elitism(agents_population, train_data=None, env=None)** - This method is used when **tuning_mode** is equal to *'best_performant_elitism'*.

  This method simply calls the method **_no_elitism_or_best_performant_elitism_common()** with **preserve_best_agent** equal to *True*.

  Parameters:

  – **agents_population** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the first generation.
  – **train_data** (*BaseDataSet*, None) - This is the original **train_data** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **train_data** for the generation.
  – **env** (*BaseEnvironment*, None) - This is the original **env** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **env** for the generation.

  Returns:

  – **agents_population** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the last generation.

    This has value *None* if something went wrong in the execution of this method.

- **_pool_elitism(agents_population, train_data=None, env=None)** - This method is used when **tuning_mode** is equal to *'pool_elitism'*.

  This method selects the top **pool_size** agents from each generation and it constructs the new generation by deep copying an equal amount of times each of the **pool_size** best agents. Then we proceed like in the previous two methods:

  – The **train_data** and-or **env** are created, by repeatedly calling the method **_mutate_gather_data_and_env()**.
  – The agents are learnt and evaluated, by repeatedly calling the method **_learn_and_evaluate()**.

  Parameters:

  – **agents_population** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the first generation.
  – **train_data** (*BaseDataSet*, None) - This is the original **train_data** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **train_data** for the generation.

- **env** (*BaseEnvironment*, None) - This is the original **env** as passed in input to the method **tune()** of the *Tuner*. It is used to generate the new **env** for the generation.

Returns:

- **agents_population** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the last generation.

  This has value *None* if something went wrong in the execution of this method.

- **tune(train_data=None, env=None)** - This method calls the *base* method **tune()** implemented in the Class *Tuner*. Then we check the result of such call:

  - If what we get is *None*, it means that such method did not return anything, and so we are free to proceed.
  - Else we return immediately the results we got passed down from such method.

  Now based on the value of **tuning_mode** we call the appropriate method:

  - If *'no_elitism'* is selected then the method **_no_elitism()** is called.
  - If *'best_performant_elitism'* is selected then the method **_best_performant_elitism()** is called.
  - If *'pool_elitism'* is selected then the method **_pool_elitism()** is called.

  After this we extract the best agent overall, according to the **eval_metric**. This best agent is saved to a *pickle* file.

  Now if **create_explanatory_heatmap** is equal to *True*, the heatmap explaining the impact of the hyper-parameters on the evaluation of **block_to_opt** is created by calling the method **create_explanatory_heatmap_hyperparameters(**

  Finally the best agent and its evaluation are returned.

  Parameters:

  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - If everything went smoothly we return: **best_agent** (*Block*), **best_agent_eval** (*float*) - These are the best agent and its evaluation.
  - If there was an error we return: *None*, *None*.

- **_mutate(agent, first_mutation=False)** - This method mutates the hyper-parameters of the provided **agent**. Precisely:

  - The method **get_params()** of the **agent** is called to extract its hyper-parameters.
  - We loop over all the hyper-parameters and we mutate those that have the member **to_mutate** equal to *True*.

    A mutation of a specific hyper-parameter takes place with probability **prob_point_mutation**.
  - We call the method **set_params()** of the **agent**: this way the new hyper-parameters are set.

  Parameters:

  - **agent** (*Block*) - This is the agent whose hyper-parameters needs to be mutated. It must be an object of a Class inheriting from the Class *Block*.

- **first_mutation** (*bool*, False) - This is a boolean and it used to decide how to perform the mutation. This is going to be passed down to the method **mutate()** of a specific hyper-parameter. To understand precisely the impact of this parameter see the documentation about the method **mutate()** of the Classes *Real* and *Integer*.

Returns:

- **agent** (*Block*) - This is the mutated agent.

  This has value *None* if something went wrong in the execution of this method.

- **_evaluate(agent_res, agent=None, train_data=None, env=None)** - This method evaluates the provided **agent_res** on the provided **train_data** and-or **env**.

  This method simply calls the method **evaluate()** of the **eval_metric** of the *Tuner*.

  Parameters:

  - **agent_res** (*BlockOutput*) - This is the output block produced by the call of the method **learn()** of the **agent**.
  - **agent** (*Block*, None) - This is the agent that was learnt.
  - **train_data** (*BaseDataSet*, None) - This must be an object of a Class inheriting from the Class *BaseDataSet*.
  - **env** (*BaseEnvironment*, None) - This must be an object of a Class inheriting from the Class *BaseEnvironment*.

  Returns:

  - **tmp_single_agent_eval** (*float*) - This is the evaluation of the provided **agent_res**.

- **_select(agents_pop)** - This method selects an agent from the current population: this agent will be passed onto the next generation. How is the agent selected?

  - A random subset of 3 agents is selected from **agents_pop**.
  - The best agent out of the three is selected.

  Parameters:

  - **agents_pop** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents the previous generation.

  Returns:

  - **selected_ag** (*Block*) - This is the selected agent that will be mutated and be part of the new generation.

- **_evaluate_a_generation(gen)** - This method orders the agents of the provided **gen** according to their evaluation (which is contained in the member **block_eval**).

  From the **gen** I extract the evaluation of the agents and then I use *numpy.argsort()*. In doing this I take into consideration whether the metric is to be maximised, or minimised.

  Parameters:

  - **gen** (*list*) - This is a list of agents, namely of objects of a Class inheriting from the Class *Block*. This represents a generation.

  Returns:

  - **best_agent** (*Block*) - This is the best agent.
  - **best_agent_eval** (*float*) - This is the best agent evaluation.

28 Further Details

The following are some facts that might be useful for using this library:

On Python version: You must use Python3.7 or above.

On policies: A policy *HyperParameter* object cannot have **to_mutate** equal to *True*: policies cannot be mutated. You can always create a *AutoModelGeneration* blocks and compare two different policies, while keeping the other things constant.

Note that not all RL algorithms work with all possible policies: to see which works with which see MushroomRL documentation. For example in deep actor critic method you can only use a *TorchPolicy*.

The **regressor_type** of a policy (and so of *ModelGeneration* blocks) has to be *'generic_regressor'* when the action space is continuous (i.e: Box).

On metrics: The Classes *Metric* that are used with *ModelGeneration* blocks must have two members: **eval_mean** and **eval_var**. These are used in the method **learn()** of the Classes *OfflineRLPipeline* and *OnlineRLPipeline* to construct the **policy_eval**.

To fill in **eval_metric** for block for which you do not care-need about the evaluation you can use the Class *SomeSpecificMetric*.

On blocks: If you follow the structure of the implemented blocks it is possible to implement any block and use it together with all the other things implemented in this library.

If you create a new tuner you can create a new Class inheriting from the Class *Tuner*, and then you can create an *AutoModelGeneration* block using your new tuner. You can then compare it with existing tuners.

If you create an automatic block make sure that the **block_to_opt** have the same metric: it does not make sense to rank them on different metrics.

On pipelines: The evaluation metric of an RL pipeline must be equal to the evaluation metric of the last block contained in the pipeline.

If you create a *AutoRLPipeline* block:

- Make sure that the pipelines have the same metric: it does not make sens to rank them on different metrics.
- You can have an automatic block (e.g: a *AutoModelGeneration* block): this is not going to be mutated as **is_parametrised** is equal to *False*.
- You must pick the task type: online or offline by setting **online_task** either to *True* or *False*.

On hyper-parameters:

- The hyper-parameters of all blocks must be specified as *HyperParameter* objects.
- At the moment the parameters of the data generation blocks used in feature engineering blocks can't be tuned.

On environments:

- MushroomRL dataset need *numpy.array* for all elements, except for the reward, hence even if the action space is *Discrete* we need a *numpy.array* for the action. Moreover MushroomRL policies predict numpy.array([action]) even if the action space is discrete. It is for this reason also that in *DataGenerationRandomUniformPolicy* passes numpy.array([action]) to all environments when extracting datasets.

  Thus the environment must assume to use np.array([action]), even if it is a Discrete Action space. To go around this you can just extract the number at the top of the method **step()** of your environment.

- All methods **reset** in all environments must have the signature described in the documentation of the Class *BaseEnvironment* otherwise MushroomRL RL algorithms are going to fail!

- All environments must inherit from the Class *BaseEnvironment*, else they cannot be used in this library. Moreover you must use the spaces *Box* and *Discrete* from MushroomRL and not those from OpenAI gym.

  No other space types are supported due to limitation of the current MushroomRL release.

- To be sure not to mess up parallel computation the method **seed()** should call the method **set_local_prng()** and also all source of randomness in the environment should use the **local_prng** provided by the *AbstractUnit*. This is because *numpy.random* is not safe to use on concurrent threads or processes.
  cf. https://numpy.org/doc/stable/reference/random/parallel.html
  cf. https://albertcthomas.github.io/good-practices-random-number-generators/

  Moreover do not use *numpy.random* for sampling in an environment but use the **local_prng** provided by the *AbstractUnit*.

  This is very important indeed in the Classes *Metric* and in some *DataGeneration* Classes the method **set_local_prng()** of the environments is called while the method **seed()** of the environment is never called!

  You should seed the environment yourself upon creation of the environment object!

On performance:

- If the evaluation seems slow you may not be using the right metric. If you are using a RL algorithm with an Epsilon Greedy policy make sure to set **batch_eval** equal to *True* in the *DiscountedReward* metric.

  Why is the **non_batch_eval** slow? Because the method **draw_action()** of object of a Class inheriting from the Class *mushroom_rl.policy.Policy* only accept in input a single sample, therefore to predict 1000 samples we need to call 1000 times the method **draw_action()**.

  Instead with **batch_eval** we can call just once the method predict.

- Make sure not to mix up too many parallelisation techniques: for example in python a fork of a fork is not possible so if you set two nested blocks to perform multi-processing this is not going to work. Moreover the fact that it is not going to work can be silent: the program may just freeze.

  Moreover keep in mind that python has the GIL therefore only one thread at the time has control of the interpreter, as such if you perform multi-threading on *pure* python is going to be slower. This is not the case for python wrapper libraries for C or C++ code (e.g: PyTorch, xgboost).
  cf. https://stackoverflow.com/questions/1912557/a-question-on-python-gil

  On this note PyTorch only allows multi-threading therefore in deep actor critic methods, where PyTorch is used, specifying **job_type** is useless.

On the method: set_params() - When setting new parameters, for any block, all the previous parameters are lost: this is to ease the tuning procedure.

Therefore you can never partially specify hyper-parameters.

On algo_params_upon_instantiation: If **algo_params** is *None* then this is set to a structured default dictionary. Now if you stop the method **learn()** without **algo_params_upon_instantiation** you would get an error since now **algo_params** is a structured dictionary but the method **set_params()** expects a flat dictionary.

The same reasoning holds for **tuner_blocks_dict_upon_instantiation**.

Things to consider when extending the library and library code styles:

- Methods starting with underscore are supposed to be private (as it is practice in python). This is however just a convention.
- Check type of a parameter also in the initialiser and not just in setter and getter methods.
- When reporting logs and messages write members and class names between single quotes like so: \'train_data\'
- When calling a method use always keyword arguments and not positional arguments, that is: write also the name of method parameter, as in the prototype of the method, and not just the value passed to such method.
- Keep each row less than 130 characters.
- In comments and messages write the word Class with a capital C.
- Use as little as possible the raising of exception and raise the proper type of exceptions.
- When calling methods **get_params()** check that you get something that is not *None*.
- When calling methods **set_params()** you have to set also **params_upon_instantiation**.
- Set **quiet** equal to *True* in MushroomRL **agents** and **core** if you were to add a new wrapper Class for new algorithms.
- Sometimes I favour repetition of code if that helps avoiding increasing the cyclomatic complexity.

29 Known compatibility issues

- Completely reproducible results are not guaranteed in PyTorch.
  cf. https://pytorch.org/docs/stable/notes/randomness.html
- MushroomRL has no ****kwargs** in the method **reset()** and there is nothing in the *mushroom_rl.core.Core* Class that could make use of them hence I remove them from the method **reset()** of *BaseEnvironment*.
- If you use a server to run a script, with Python3.8, and then you use a PC with Python3.7 to checkout the results it will not be possible: You cannot load a pickled file in Python3.7 that was pickled by Python3.8.

  Indeed there is a new attribute of **inspect** added in Python3.8 called **co_posonlyargcount**: in Python3.8 this attribute is serialised, but when de-serialising Python3.7 does not expect this value and it raises an exception.

  You can see the differences in the definitions here:

  - Python3.8 and above: https://docs.python.org/3/library/inspect.html
  - Python 3.7: https://docs.python.org/3.7/library/inspect.html
- For pickling *cloudpickle* is used and not *dill* since *dill* can not serialise ABC data objects in Python greater than 3.7.
- A RL Algorithm may return NaN which may break the environment. See:
  - https://github.com/hill-a/stable-baselines/issues/340
  - https://stable-baselines.readthedocs.io/en/master/guide/checking_nan.html

  This means that a trial in a tuner may fail for some strange hyper-parameters configurations.

  I check this and catch the exception: in the Class *TunerOptuna* exceptions are handled natively.

  In the Class *TunerGenetic* I pass onto the next generation the agent with the hyper-parameters that returned NaN and I assign to it a $\pm\infty$ evaluation ($+\infty$ if the metric should be minimised, $-\infty$ if the metric should be maximised). It may happen that this agent is going to be picked to pass onto the next generation: indeed tournament selection is performed. This is not an issue indeed when handling the exception I also mutate its hyper-parameters using **first_mutation=True**.
- When using *'CMA-ES'* in *TunerOptuna* it may raise a warning telling yout that is using independent sampling because *'CMA-ES'* does not support dynamic search space and-or categorical hyper-parameters.

  This is a bug in Optuna.
- You cannot pickle a Class that is not Global: therefore for *ModelGeneration* blocks where I have the method **_default_network()** it will not be possible to pickle that network class hence it will not be possible to perform multi-processing.

  Note that this is not an issue for multi-threading!
- MushroomRL is directly compatible only with sklearn. It is also compatible with xgboost (but this is due to xgboost design): this is rather an exception than a rule. Indeed it does not work with catboost.

  This can be seen in lines $69-71$ of the following link:
  https://github.com/MushroomRL/mushroom-rl/blob/dev/mushroom_rl/approximators/regressor.py

  The problem is that **input_shape** is passed down to the approximator: xgboost can filter it, but catboost cannot. Therefore you will not be able to use catboost out-of-the-box.

You can always wrap the CatBoostRegressor Class to solve this issue.

- If you extract from a *Tuner* object the **eval_metric** this is different from the **eval_metric** that yu can extract from the **tuner_blocks_dict** of an automatic block.

  This is not a bug, it is simply because in the method **pre_learn_check()** of automatic blocks I deep copy to **tuner_blocks_dict** the value of **tuner_blocks**d**ict_upon_instantiation**.

  The same holds for non automatic blocks in which I deep copy to **algo_params** the value of **algo_params_upon_instantiation**.
