NiaAML

Release 1.1.11

Luka Pečnik

Jan 20, 2023

USER DOCUMENTATION

NiaAML is an automated machine learning Python framework based on nature-inspired algorithms for optimization. The name comes from the automated machine learning method of the same name [1]. Its goal is to efficiently compose the best possible classification pipeline for the given task using components on the input. The components are divided into three groups: feature seletion algorithms, feature transformation algorithms and classifiers. The framework uses nature-inspired algorithms for optimization to choose the best set of components for the classification pipeline on the output and optimize their parameters. We use NiaPy framework for the optimization process which is a popular Python collection of nature-inspired algorithms. The NiaAML framework is easy to use and customize or expand to suit your needs.

- **Free software:** MIT license
- **Github repository:** https://github.com/lukapecnik/NiaAML
- **Python versions:** 3.6.x, 3.7.x, 3.8.x

The main documentation is organized into a couple of sections:

- *User Documentation*
- *Developer Documentation*
- *About*

CHAPTER

ONE

GETTING STARTED

This section is going to show you how to use the NiaAML framework. First install NiaAML package using the following command:

```
pip3 install niaaml
```

After the successful installation you are ready to run your first example.

1.1 Basic example

Create a new file, with name, for example *my_first_pipeline.py* and paste in the code below.

```
from niaaml import PipelineOptimizer, Pipeline
from niaaml.data import BasicDataReader
import numpy

# dummy random data
data_reader = BasicDataReader(
    x=numpy.random.uniform(low=0.0, high=15.0, size=(50, 3)),
    y=numpy.random.choice(['Class 1', 'Class 2'], size=50)
)

pipeline_optimizer = PipelineOptimizer(
    data=data_reader,
    classifiers=['AdaBoost', 'Bagging', 'MultiLayerPerceptron', 'RandomForest',
↪'ExtremelyRandomizedTrees', 'LinearSVC'],
    feature_selection_algorithms=['SelectKBest', 'SelectPercentile',
↪'ParticleSwarmOptimization', 'VarianceThreshold'],
    feature_transform_algorithms=['Normalizer', 'StandardScaler']
)
pipeline = pipeline_optimizer.run('Accuracy', 15, 15, 300, 300, 'ParticleSwarmAlgorithm',
↪ 'ParticleSwarmAlgorithm')
```

As you can see, pipeline components, fitness function and optimization algorithms are always passed into pipeline optimization using their class names. The example below uses the Particle Swarm Algorithm as the optimization algorithm. You can find a list of all available algorithms in the NiaPy's documentation. Now you can run it using the command `python3 my_first_pipeline.py`. The code currently does not do much, but we can save our pipeline to a file so we can use it later or save a user-friendly representation of it to a text file. You can choose one or both of the scenarios by adding the code below.

```
pipeline.export('pipeline.ppln')
pipeline.export_text('pipeline.txt')
```

If you want to load and use the saved pipeline later, you can use the following code.

```
from niaaml import Pipeline
import pandas

loaded_pipeline = Pipeline.load('pipeline.ppln')

# some features (can be loaded using DataReader object instances)
x = pandas.DataFrame([[0.35, 0.46, 5.32], [0.16, 0.55, 12.5]])
y = loaded_pipeline.run(x)
```

The framework also supports the original version of optimization process where the components selection and hyperparameter optimization steps are combined into one. You can replace the ``run`` method with the following code.

```
pipeline = pipeline_optimizer.run_v1('Accuracy', 15, 400, 'ParticleSwarmAlgorithm')
```

This is a very simple example with dummy data. It is only intended to give you a basic idea on how to use the framework. **NiaAML supports numerical and categorical features.**

Find more examples here

1.2 Components

In the following sections you can see a list of currently implemented components divided into groups: classifiers, feature selection algorithms and feature transformation algorithms. At the end you can also see a list of currently implemented fitness functions for the optimization process. Values in parentheses are associated names.

1.2.1 Classifiers

- Adaptive Boosting (AdaBoost),
- Bagging (Bagging),
- Extremely Randomized Trees (ExtremelyRandomizedTrees),
- Linear SVC (LinearSVC),
- Multi Layer Perceptron (MultiLayerPerceptron),
- Random Forest Classifier (RandomForest),
- Decision Tree Classifier (DecisionTree),
- K-Neighbors Classifier (KNeighbors),
- Gaussian Process Classifier (GaussianProcess),
- Gaussian Naive Bayes (GaussianNB),
- Quadratic Discriminant Analysis (QuadraticDiscriminantAnalysis).

1.2.2 Feature Selection Algorithms

- Select K Best (SelectKBest),
- Select Percentile (SelectPercentile),
- Variance Threshold (VarianceThreshold).

Nature-Inspired

- Bat Algorithm (BatAlgorithm),
- Differential Evolution (DifferentialEvolution),
- Self-Adaptive Differential Evolution (jDEFSTH),
- Grey Wolf Optimizer (GreyWolfOptimizer),
- Particle Swarm Optimization (ParticleSwarmOptimization).

1.2.3 Feature Transformation Algorithms

- Normalizer (Normalizer),
- Standard Scaler (StandardScaler),
- Maximum Absolute Scaler (MaxAbsScaler),
- Quantile Transformer (QuantileTransformer),
- Robust Scaler (RobustScaler).

1.2.4 Fitness Functions based on

- Accuracy (Accuracy),
- Cohen's kappa (CohenKappa),
- F1-Score (F1),
- Precision (Precision).

1.2.5 Categorical Feature Encoders

- One-Hot Encoder (OneHotEncoder).

1.2.6 Feature Imputers

- Simple Imputer (SimpleImputer).

1.2.7 Optimization Algorithms

For the list of available optimization algorithms please see the NiaPy's documentation.

1.3 Optimization Process And Parameter Tuning

In NiaAML there are two types of optimization. Goal of the first type is to find an optimal set of components (feature selection algorithm, feature transformation algorithm and classifier). The next step is to find optimal parameters for the selected set of components and that is a goal of the second type of optimization. Each component has an attribute **_params**, which is a dictionary of parameters and their possible values.

```
self._params = dict(
    n_estimators = ParameterDefinition(MinMax(min=10, max=111), np.uint),
    algorithm = ParameterDefinition(['SAMME', 'SAMME.R'])
)
```

An individual in the second type of optimization is a real-valued vector that has a size equal to the sum of number of keys in all three dictionaries (classifier's _params, feature transformation algorithm's _params and feature selection algorithm's _params) and a value of each dimension is in range [0.0, 1.0]. The second type of optimization maps real values from the individual's vector to those parameter definitions in the dictionaries. Each parameter's value can be defined as a range or array of values. In the first case, a value from vector is mapped from one iterval to another and in the second case, a value from vector falls into one of the bins that represent an index of the array that holds possible parameter's values.

Let's say we have a classifier with 3 parameters, feature selection algorithm with 2 parameters and feature transformation algorithm with 4 parameters. Size of an individual in the second type of optimization is 9. Size of an individual in the first type of optimization is always 3 (1 classifier, 1 feature selection algorithm and 1 feature transform algorithm).

In some cases we may want to tune a parameter that needs additional information for setting its range of values, so we cannot set the range in the initialization method. In that case we should set its value in the dictionary to None and define it later in the process. The parameter will be a part of parameter tuning process as soon as we define its possible values. For example, see the implementation of *niaaml.preprocessing.feature_selection.SelectKBest* and its parameter **k**.

CHAPTER

TWO

CHANGELOG

2.1 1.1.10 (2022-08-17)

Full Changelog

Closed issues:

- Publish to PyPI #73

Merged pull requests:

- Update dependencies #76 (firefly-cpp)

2.2 1.1.9 (2022-05-25)

Full Changelog

Closed issues:

- Test with python-scikit-learn 1.1.0 is not passing #71

Merged pull requests:

- Bump version #75 (firefly-cpp)
- Do not package the tests #74 (firefly-cpp)

2.3 1.1.8 (2022-05-24)

Full Changelog

Merged pull requests:

- Removed deprecated sklearn warnings #72 (zStupan)
- Update README.md #70 (firefly-cpp)
- Run tests with github action #68 (lukapecnik)
- docs: add lukapecnik as a contributor for infra #67 (allcontributors[bot])
- docs: add musicinmybrain as a contributor for code, infra #66 (allcontributors[bot])
- docs: add zStupan as a contributor for code #65 (allcontributors[bot])

2.4 1.1.7 (2022-02-21)

Full Changelog

Closed issues:

- Update to the latest niapy stable release #62
- Example file pipeline.ppln is out of date #60
- np.int is a deprecated alias #56
- Remove setup.py file #54

Merged pull requests:

- Update niapy #64 (lukapecnik)
- Update README.md #63 (firefly-cpp)
- Eschew deprecated numpy aliases for builtins #61 (musicinmybrain)
- Corrected dependencies in CONTRIBUTING #59 (firefly-cpp)
- Do not install text files in site-packages/ #58 (musicinmybrain)
- Use the latest upstream's release for niapy #57 (firefly-cpp)
- Add reference links #55 (firefly-cpp)
- Add link to the API #53 (firefly-cpp)

2.5 1.1.6 (2021-06-27)

Full Changelog

Closed issues:

- Upgrade to the latest niapy release #51

Merged pull requests:

- Update to the latest niapy version and fix docs build warnings #52 (zStupan)

2.6 1.1.5 (2021-06-01)

Full Changelog

Merged pull requests:

- Citation details #50 (firefly-cpp)

2.7 1.1.4 (2021-05-24)

Full Changelog

Closed issues:

- Unable to install with conda #47

2.8 1.1.3 (2021-05-23)

Full Changelog

2.9 1.1.2 (2021-05-19)

Full Changelog

Merged pull requests:

- Update niapy dependency #49 (lukapecnik)

2.10 1.1.1 (2021-03-09)

Full Changelog

Merged pull requests:

- logo [ci skip] #48 (lukapecnik)
- Fedora package pushed to stable #46 (firefly-cpp)
- Restructuring of paper and editorial changes #45 (adi3)
- Corrections of paper #44 (firefly-cpp)

2.11 1.1.1rc2 (2020-12-22)

Full Changelog

2.12 1.1.1rc1 (2020-12-22)

Full Changelog

Merged pull requests:

- Original NiaAML method support [ci skip] #43 (lukapecnik)

2.13 1.1.0 (2020-12-16)

Full Changelog

2.14 1.0.0rc7 (2020-12-14)

Full Changelog

Closed issues:

- References #40

Merged pull requests:

- Paper update #42 (firefly-cpp)
- minor corrections in examples #39 (firefly-cpp)

2.15 1.0.0rc6 (2020-12-12)

Full Changelog

Closed issues:

- Conda package #34

Merged pull requests:

- Additional features #38 (lukapecnik)
- Conda fix [ci skip] #37 (lukapecnik)
- Orcid identifiers added #36 (firefly-cpp)

2.16 1.0.0rc5 (2020-12-11)

Full Changelog

Closed issues:

- Installation problems #31

Merged pull requests:

- Conda support #35 (lukapecnik)

2.17 1.0.0rc4 (2020-12-10)

Full Changelog

Merged pull requests:

- Python 3.6 support #33 (lukapecnik)
- First version of paper #32 (firefly-cpp)

2.18 1.0.0rc3 (2020-12-10)

Full Changelog

2.19 1.0.0rc2 (2020-12-08)

Full Changelog

Merged pull requests:

- feature missing values imputation #30 (lukapecnik)
- README updated #29 (firefly-cpp)
- Readme update, encoder check for type int64 #28 (lukapecnik)
- Markdown and docs #27 (lukapecnik)

2.20 1.0.0rc1 (2020-12-06)

Full Changelog

Merged pull requests:

- Added support for categorical features #26 (lukapecnik)

2.21 0.1.4 (2020-12-05)

Full Changelog

Merged pull requests:

- remove 10-fold cross validation from benchmark, critical bug fix #25 (lukapecnik)
- run all experiments at once #24 (firefly-cpp)

2.22 0.1.3 (2020-12-04)

Full Changelog

2.23 0.1.3a1 (2020-12-01)

Full Changelog

Merged pull requests:

- 0.1.3a1 pre-release python 3.7 compatibility #23 (lukapecnik)
- Fixes and additions #22 (lukapecnik)

2.24 0.1.2 (2020-11-30)

Full Changelog

Implemented enhancements:

- On the use of unittest #2

Closed issues:

- Description of examples #16

Merged pull requests:

- readme.rst fix, pipeline fix #21 (lukapecnik)
- Travis ci integration #20 (lukapecnik)
- badges and readme update #19 (lukapecnik)

2.25 0.1.2a1 (2020-11-29)

Full Changelog

Closed issues:

- Information about hyperparameter tuning #15
- CHANGELOG #14
- Examples #13

Merged pull requests:

- Unittests, examples' description, references added to docs #17 (lukapecnik)

2.26 0.1.1 (2020-11-28)

Full Changelog

Closed issues:

- Installation instructions #11

Merged pull requests:

- Contributors table added #12 (lukapecnik)

2.27 0.1.0 (2020-11-27)

Full Changelog

Implemented enhancements:

- CSV Data Reader class #3

Closed issues:

- A non-functional demo could be written #4

Merged pull requests:

- Framework improvements, docs initialization and readme #10 (lukapecnik)
- Pipeline methods implementation #9 (lukapecnik)
- Pipeline optimizer progress #8 (lukapecnik)
- Implementation of jDEFSTH algorithm for feature selection #7 (firefly-cpp)
- refactoring and variance threshold feature selection implementation #6 (lukapecnik)
- NiaPy dependency added #5 (firefly-cpp)

CHAPTER

THREE

INSTALLATION

3.1 Setup development environment

3.1.1 Requirements

- Poetry: https://python-poetry.org/docs/

After installing Poetry and cloning the project from GitHub, you should run the following command from the root of the cloned project:

```
$ poetry install
```

All of the project's dependencies should be installed and the project ready for further development. **Note that Poetry creates a separate virtual environment for your project.**

3.1.2 Development dependencies

List of NiaAML's dependencies:

Package	Version	Platform
numpy	^1.19.1	All
scikit-learn	^0.23.2	All
NiaPy	^2.0.0rc11	All
pandas	^1.1.4	All

List of development dependencies:

Package	Version	Platform
sphinx	^3.3.1	Any
sphinx-rtd-theme	^0.5.0	Any
coveralls	^2.2.0	Any

CHAPTER

FOUR

TESTING

Before making a pull request, if possible provide tests for added features or bug fixes.

We have an automated building system which also runs all of provided tests. In case any of the test cases fails, we are notified about failing tests. Those should be fixed before we merge your pull request to master branch.

For the purpose of checking if all test are passing localy you can run following command:

```
$ poetry run coverage run --source=niaaml -m unittest discover -b
```

If all tests passed running this command it is most likely that the tests would pass on our build system too.

CHAPTER

FIVE

DOCUMENTATION

To locally generate and preview documentation run the following command in the project root folder:

```
$ poetry run sphinx-build ./docs ./docs/_build
```

If the build of the documentation is successful, you can preview the documentation in the docs/_build folder by clicking the `index.html` file.

CHAPTER

SIX

API

This is the NiaAML API documentation, auto generated from the source code.

6.1 niaaml

class niaaml.**Factory**(***kwargs*)

Bases: object

Base class with string mappings to entities.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
_entities (Dict[str, any]): Dictionary to map from strings to an instance of anything.

get_name_to_classname_mapping()

Get dictionary of user-friendly name to class name mapping.

Returns:
dict: Dictionary of user-friendly name to class name mapping.

get_result(*name*)

Get the resulting entity.

Arguments:
name (str): String that represents the entity.

Returns:
any: Entity according to the given name.

class niaaml.**Logger**(*verbose=False*, *output_file=None*, ***kwargs*)

Bases: object

Class for logging throughout the framework.

Date:
2020

Author:
Luka Pečnik

License:
MIT

`log_optimization_error`(*text*)

Log optimization error message.

`log_pipeline`(*text*)

Log pipeline info message.

`log_progress`(*text*)

Log progress message.

`class` niaaml.`MinMax`(*min*, *max*)

Bases: `object`

Class for ParameterDefinition's value property.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
min (float): Minimum number (inclusive). max (float): Maximum number (exclusive).

See Also:

- `niaaml.utilities.ParameterDefinition`

`class` niaaml.`OptimizationStats`(*predicted*, *expected*, ***kwargs*)

Bases: `object`

Class that holds pipeline optimization result's statistics. Includes accuracy, precision, Cohen's kappa and F1-score.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
_accuracy (float): Calculated accuracy. _precision (float): Calculated precision. _cohen_kappa (float): Calculated Cohen's kappa. _f1_score (float): Calculated F1-score.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.**ParameterDefinition**(*value*, *param_type=None*)

Bases: object

Class for PipelineComponent parameters definition.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
value (any): Array of possible parameter values or instance of MinMax class. param_type (numpy.dtype): Selection output data type.

See Also:

- niaaml.pipeline_component.PipelineComponent
- niaaml.utilities.MinMax

class niaaml.**Pipeline**(***kwargs*)

Bases: object

Classification pipeline defined by optional preprocessing steps and classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
__feature_selection_algorithm (Optional[FeatureSelectionAlgorithm]): Feature selection algorithm implementation. __feature_transform_algorithm (Optional[FeatureTransformAlgorithm]): Feature transform algorithm implementation. __classifier (Classifier): Classifier implementation. __selected_features_mask (Iterable[bool]): Mask of selected features during the feature selection process. __best_stats (OptimizationStats): Statistics of the most successful setup of parameters. __categorical_features_encoders (Dict[FeatureEncoder]): Instances of FeatureEncoder for all categorical features. __imputers (Dict[Imputer]): Dictionary of instances of Imputer for all columns that contained missing values during optimization process. __logger (Logger): Logger instance.

export(*file_name*)

Exports Pipeline object to a file for later use. Extension is added if not present.

Arguments:
file_name (str): Output file name.

export_text(*file_name*)

Exports Pipeline object to a user-friendly text file. Extension is added if not present.

Arguments:
file_name (str): Output file name.

`get_classifier`()

Get deep copy of the classifier.

Returns:
Classifier: Instance of the Classifier object.

`get_feature_selection_algorithm`()

Get deep copy of the feature selection algorithm.

Returns:
FeatureSelectionAlgorithm: Instance of the FeatureSelectionAlgorithm object.

`get_feature_transform_algorithm`()

Get deep copy of the feature transform algorithm.

Returns:
FeatureTransformAlgorithm: Instance of the FeatureTransformAlgorithm object.

`get_logger`()

Get logger.

Returns:
Logger: Instance of the Logger object.

`get_stats`()

Get optimization statistics.

Returns:
OptimizationStats: Instance of the OptimizationStats object.

`static load`(*file_name*)

Loads Pipeline object from a file.

Returns:
Pipeline: Loaded Pipeline instance.

`optimize`(*x*, *y*, *population_size*, *number_of_evaluations*, *optimization_algorithm*, *fitness_function*)

Optimize pipeline's hyperparameters.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array. population_size (uint): Number of individuals in the optimization process. number_of_evaluations (uint): Number of maximum evaluations. optimization_algorithm (str): Name of the optimization algorithm to use. fitness_function (str): Name of the fitness function to use.

Returns:
float: Best fitness value found in optimization process.

`run`(*x*)

Runs the pipeline.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes of the samples in the x array.

`set_categorical_features_encoders`(*value*)

Set categorical features' encoders.

set_classifier(*value*)

Set classifier.

set_feature_selection_algorithm(*value*)

Set feature selection algorithm.

set_feature_transform_algorithm(*value*)

Set feature transform algorithm.

set_imputers(*value*)

Set imputers.

set_selected_features_mask(*value*)

Set selected features mask.

set_stats(*value*)

Set stats.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

to_string_slim()

Slim user friendly representation of the object.

Returns:
str: Slim user friendly representation of the object.

class niaaml.**PipelineComponent**(***kwargs*)

Bases: `object`

Class for implementing pipeline components.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
Name (str): Name of the pipeline component. _params (Dict[str, ParameterDefinition]): Dictionary of components's parameters with possible values. Possible parameter values are given as an instance of the ParameterDefinition class.

See Also:

- `niaaml.utilities.ParameterDefinition`

Name = None

get_params_dict()

Return parameters definition dictionary.

set_parameters(***kwargs*)

Set the parameters/arguments of the pipeline component.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.PipelineOptimizer(**kwargs)

Bases: object

Optimization task that finds the best classification pipeline according to the given input.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
__data (DataReader): Instance of any DataReader implementation. __feature_selection_algorithms (Optional[Iterable[str]]): Array of names of possible feature selection algorithms. __feature_transform_algorithms (Optional[Iterable[str]]): Array of names of possible feature transform algorithms. __classifiers (Iterable[Classifier]): Array of names of possible classifiers. __categorical_features_encoder (str): Name of the encoder used for categorical features. __categorical_features_encoders (Dict[FeatureEncoder]): Actual instances of FeatureEncoder for all categorical features. __imputer (str): Name of the imputer used for features that contain missing values. __imputers (Dict[Imputer]): Actual instances of Imputer for all features that contain missing values. __logger (Logger): Logger instance.

get_classifiers()

Get classifiers.

Returns:
Iterable[str]: Classifier names.

get_data()

Get data.

Returns:
DataReader: Instance of DataReader object.

get_feature_selection_algorithms()

Get feature selection algorithms.

Returns:
Iterable[str]: Feature selection algorithm names or None.

get_feature_transform_algorithms()

Get feature transform algorithms.

Returns:
Iterable[str]: Feature transform algorithm names or None.

get_logger()

Get logger.

Returns:
Logger: Logger instance.

run(*fitness_name*, *pipeline_population_size*, *inner_population_size*, *number_of_pipeline_evaluations*, *number_of_inner_evaluations*, *optimization_algorithm*, *inner_optimization_algorithm=None*)

Run classification pipeline optimization process.

Arguments:
fitness_name (str): Name of the fitness class to use as a function. pipeline_population_size (uint): Number of pipeline individuals in the optimization process. inner_population_size (uint): Number of individuals in the hiperparameter optimization process. number_of_pipeline_evaluations (uint): Number of maximum evaluations. number_of_inner_evaluations (uint): Number of maximum inner evaluations. optimization_algorithm (str): Name of the optimization algorithm to use. inner_optimization_algorithm (Optional[str]): Name of the inner optimization algorithm to use. Defaults to the optimization_algorithm argument.

Returns:
Pipeline: Best pipeline found in the optimization process.

run_v1(*fitness_name*, *population_size*, *number_of_evaluations*, *optimization_algorithm*)

Run classification pipeline optimization process according to the original NiaAML paper.

Reference:
Fister, Iztok, Milan Zorman, and Dušan Fister. "Continuous Optimizers for Automatic Design and Evaluation of Classification Pipelines." Frontier Applications of Nature Inspired Computation. Springer, Singapore, 2020. 281-301.

Arguments:
fitness_name (str): Name of the fitness class to use as a function. population_size (uint): Number of individuals in the optimization process. number_of_evaluations (uint): Number of maximum evaluations. optimization_algorithm (str): Name of the optimization algorithm to use.

Returns:
Pipeline: Best pipeline found in the optimization process.

niaaml.**get_bin_index**(*value*, *number_of_bins*)

Gets index of value's bin. Value must be between 0.0 and 1.0.

Arguments:
value (float): Value to put into bin. number_of_bins (uint): Number of bins on the interval [0.0, 1.0].

Returns:
uint: Calculated index.

6.2 niaaml.data

class niaaml.data.**BasicDataReader**(***kwargs*)

Bases: *DataReader*

Implementation of basic data reader.

Date:
2020

Author:
Luka Pečnik

License:
MIT

See Also:

- *niaaml.data.DataReader*

class niaaml.data.**CSVDataReader**(***kwargs*)

Bases: *DataReader*

Implementation of CSV data reader.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
__src (string): Path to a CSV file. __contains_classes (bool): Tells if src contains expected classification results or only features. __has_header (bool): Tells if src contains header row.

See Also:

- *niaaml.data.DataReader*

class niaaml.data.**DataReader**(***kwargs*)

Bases: `object`

Class for implementing data readers with different sources of data.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
_x (pandas.core.frame.DataFrame): Array of rows from dataset without expected classification results. _y (Optional[pandas.core.series.Series]): Array of encoded expected classification results.

get_x()

Get value of _x.

Returns:
pandas.core.frame.DataFrame: Array of rows from dataset without expected classification results.

get_y()

Get value of _y.

Returns:
pandas.core.series.Series: Array of encoded expected classification results.

set_x(*value*)

Set the value of _x.

set_y(*value*)

Set the value of _y.

6.3 niaaml.classifiers

class niaaml.classifiers.**AdaBoost**(***kwargs*)

Bases: *Classifier*

Implementation of AdaBoost classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:

Y. Freund, R. Schapire, "A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting", 1995.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

See Also:

- *niaaml.classifiers.Classifier*

Name = 'AdaBoost'

fit(*x*, *y*, ***kwargs*)

Fit AdaBoost.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

predict(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.classifiers.**Bagging**(***kwargs*)

Bases: *Classifier*

Implementation of bagging classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:

L. Breiman, "Bagging predictors", Machine Learning, 24(2), 123-140, 1996.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

See Also:

- *niaaml.classifiers.Classifier*

`Name = 'Bagging'`

`fit`(*x*, *y*, ***kwargs*)

Fit Bagging.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

`predict`(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

`set_parameters`(***kwargs*)

Set the parameters/arguments of the algorithm.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

`class` niaaml.classifiers.`Classifier`(***kwargs*)

Bases: `PipelineComponent`

Class for implementing classifiers.

Date:
2020

Author:
Luka Pečnik

License:
MIT

See Also:

- `niaaml.pipeline_component.PipelineComponent`

fit(*x*, *y*, ***kwargs*)

Fit implemented classifier.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

predict(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

class niaaml.classifiers.**ClassifierFactory**(***kwargs*)

Bases: `Factory`

Class with string mappings to classifiers.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
_entities (Dict[str, Classifier]): Mapping from strings to classifiers.

See Also:

- `niaaml.utilities.Factory`

class niaaml.classifiers.**DecisionTree**(***kwargs*)

Bases: *Classifier*

Implementation of decision tree classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:

L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and Regression Trees", Wadsworth, Belmont, CA, 1984.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

See Also:

- *niaaml.classifiers.Classifier*

Name = 'Decision Tree Classifier'

fit(*x*, *y*, ***kwargs*)

Fit DecisionTree.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

predict(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.classifiers.**ExtremelyRandomizedTrees**(***kwargs*)

Bases: *Classifier*

Implementation of extremely randomized trees classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:

P. Geurts, D. Ernst., and L. Wehenkel, "Extremely randomized trees", Machine Learning, 63(1), 3-42, 2006.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

See Also:

- *niaaml.classifiers.Classifier*

Name = 'Extremely Randomized Trees'

fit(*x*, *y*, ***kwargs*)

Fit ExtremelyRandomizedTrees.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

`predict`(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

`set_parameters`(***kwargs*)

Set the parameters/arguments of the algorithm.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

`class` niaaml.classifiers.`GaussianNB`(***kwargs*)

Bases: *Classifier*

Implementation of gaussian Naive Bayes classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:
Murphy, Kevin P. "Naive bayes classifiers." University of British Columbia 18 (2006): 60.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

See Also:

- *niaaml.classifiers.Classifier*

`Name = 'Gaussian Naive Bayes'`

`fit`(*x*, *y*, ***kwargs*)

Fit GaussianNB.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

predict(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.classifiers.**GaussianProcess**(***kwargs*)

Bases: *Classifier*

Implementation of gaussian process classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:
Rasmussen, Carl Edward, and Hannes Nickisch. "Gaussian processes for machine learning (GPML) toolbox." The Journal of Machine Learning Research 11 (2010): 3011-3015.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifier.html

See Also:

- *niaaml.classifiers.Classifier*

Name = 'Gaussian Process Classifier'

fit(*x*, *y*, ***kwargs*)

Fit GaussianProcess.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

predict(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

`set_parameters`(***kwargs*)

Set the parameters/arguments of the algorithm.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

`class niaaml.classifiers.KNeighbors`(***kwargs*)

Bases: *Classifier*

Implementation of k neighbors classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:
"Neighbourhood Components Analysis", J. Goldberger, S. Roweis, G. Hinton, R. Salakhutdinov, Advances in Neural Information Processing Systems, Vol. 17, May 2005, pp. 513-520.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

See Also:

- *niaaml.classifiers.Classifier*

`Name = 'K Neighbors Classifier'`

`fit`(*x*, *y*, ***kwargs*)

Fit KNeighbors.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

`predict`(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

`set_parameters`(***kwargs*)

Set the parameters/arguments of the algorithm.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

`class` niaaml.classifiers.`LinearSVC`(***kwargs*)

Bases: *Classifier*

Implementation of linear support vector classification.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:
Fan, Rong-En, et al. "LIBLINEAR: A library for large linear classification." Journal of machine learning research 9.Aug (2008): 1871-1874.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

See Also:

- *niaaml.classifiers.Classifier*

`Name = 'Linear Support Vector Classification'`

`fit`(*x*, *y*, ***kwargs*)

Fit LinearSVCClassifier.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

`predict`(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

`set_parameters`(***kwargs*)

Set the parameters/arguments of the algorithm.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.classifiers.**MultiLayerPerceptron**(***kwargs*)

Bases: *Classifier*

Implementation of multi-layer perceptron classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:
Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International Conference on Artificial Intelligence and Statistics. 2010.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

See Also:

- *niaaml.classifiers.Classifier*

Name = 'Multi Layer Perceptron'

fit(*x*, *y*, ***kwargs*)

Fit MultiLayerPerceptron.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

predict(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.classifiers.**QuadraticDiscriminantAnalysis**(***kwargs*)

Bases: *Classifier*

Implementation of quadratic discriminant analysis classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:
"The Elements of Statistical Learning", Hastie T., Tibshirani R., Friedman J., Section 4.3, p.106-119, 2008.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html#sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis

See Also:

- *niaaml.classifiers.Classifier*

Name = 'Quadratic Discriminant Analysis'

fit(*x*, *y*, ***kwargs*)

Fit QuadraticDiscriminantAnalysis.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

predict(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.classifiers.**RandomForest**(***kwargs*)

Bases: *Classifier*

Implementation of random forest classifier.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:
Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 2001.

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

See Also:

- *niaaml.classifiers.Classifier*

`Name = 'Random Forest Classifier'`

`fit`(*x*, *y*, ***kwargs*)

Fit RandomForestClassifier.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify. y (pandas.core.series.Series): n classes of the samples in the x array.

Returns:
None

`predict`(*x*, ***kwargs*)

Predict class for each sample (row) in x.

Arguments:
x (pandas.core.frame.DataFrame): n samples to classify.

Returns:
pandas.core.series.Series: n predicted classes.

`set_parameters`(***kwargs*)

Set the parameters/arguments of the algorithm.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

6.4 niaaml.preprocessing

`class` niaaml.preprocessing.`PreprocessingAlgorithm`(***kwargs*)

Bases: `PipelineComponent`

Class for implementing preprocessing algorithms.

Date:
2020

Author:
Luka Pečnik

License:
MIT

See Also:

- `niaaml.pipeline_component.PipelineComponent`

6.4.1 niaaml.preprocessing.feature_selection

class niaaml.preprocessing.feature_selection.**BatAlgorithm**(***kwargs*)

Bases: *FeatureSelectionAlgorithm*

Implementation of feature selection using BA algorithm.

Date:
2020

Author:
Luka Pečnik

Reference:
The implementation is adapted according to the following article: D. Fister, I. Fister, T. Jagrič, I. Fister Jr., J. Brest. A novel self-adaptive differential evolution for feature selection using threshold mechanism . In: Proceedings of the 2018 IEEE Symposium on Computational Intelligence (SSCI 2018), pp. 17-24, 2018.

Reference URL:
http://iztok-jr-fister.eu/static/publications/236.pdf

License:
MIT

See Also:

- *niaaml.preprocessing.feature_selection.feature_selection_algorithm.FeatureSelectionAlgorithm*

Name = 'Bat Algorithm'

select_features(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
pandas.core.frame.DataFrame: Mask of selected features.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.preprocessing.feature_selection.**DifferentialEvolution**(***kwargs*)

Bases: *FeatureSelectionAlgorithm*

Implementation of feature selection using DE algorithm.

Date:
2020

Author:
Luka Pečnik

Reference:
The implementation is adapted according to the following article: D. Fister, I. Fister, T. Jagrič, I. Fister Jr., J. Brest. A novel self-adaptive differential evolution for feature selection using threshold mechanism . In: Proceedings of the 2018 IEEE Symposium on Computational Intelligence (SSCI 2018), pp. 17-24, 2018.

Reference URL:
http://iztok-jr-fister.eu/static/publications/236.pdf

License:
MIT

See Also:

- *niaaml.preprocessing.feature_selection.feature_selection_algorithm.FeatureSelectionAlgorithm*

Name = 'Differential Evolution'

select_features(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
numpy.ndarray[bool]: Mask of selected features.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.preprocessing.feature_selection.**FeatureSelectionAlgorithm**(***kwargs*)

Bases: *PreprocessingAlgorithm*

Class for implementing feature selection algorithms.

Date:
2020

Author:
Luka Pečnik

License:
MIT

See Also:

- *niaaml.preprocessing.preprocessing_algorithm.PreprocessingAlgorithm*

select_features(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
numpy.ndarray[bool]: Mask of selected features.

class niaaml.preprocessing.feature_selection.**FeatureSelectionAlgorithmFactory**(***kwargs*)

Bases: `Factory`

Class with string mappings to feature selection algorithms.

Attributes:
_entities (Dict[str, FeatureSelectionAlgorithm]): Mapping from strings to feature selection algorithms.

See Also:

- `niaaml.utilities.Factory`

class niaaml.preprocessing.feature_selection.**GreyWolfOptimizer**(***kwargs*)

Bases: *FeatureSelectionAlgorithm*

Implementation of feature selection using GWO algorithm.

Date:
2020

Author:
Luka Pečnik

Reference:
The implementation is adapted according to the following article: D. Fister, I. Fister, T. Jagrič, I. Fister Jr., J. Brest. A novel self-adaptive differential evolution for feature selection using threshold mechanism . In: Proceedings of the 2018 IEEE Symposium on Computational Intelligence (SSCI 2018), pp. 17-24, 2018.

Reference URL:
http://iztok-jr-fister.eu/static/publications/236.pdf

License:
MIT

See Also:

- *niaaml.preprocessing.feature_selection.feature_selection_algorithm.FeatureSelectionAlgorithm*

Name = 'Grey Wolf Optimizer'

select_features(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
numpy.ndarray[bool]: Mask of selected features.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.preprocessing.feature_selection.**ParticleSwarmOptimization**(***kwargs*)

Bases: *FeatureSelectionAlgorithm*

Implementation of feature selection using PSO algorithm.

Date:
2020

Author:
Luka Pečnik

Reference:
The implementation is adapted according to the following article: D. Fister, I. Fister, T. Jagrič, I. Fister Jr., J. Brest. A novel self-adaptive differential evolution for feature selection using threshold mechanism . In: Proceedings of the 2018 IEEE Symposium on Computational Intelligence (SSCI 2018), pp. 17-24, 2018.

Reference URL:
http://iztok-jr-fister.eu/static/publications/236.pdf

License:
MIT

See Also:

- *niaaml.preprocessing.feature_selection.feature_selection_algorithm.FeatureSelectionAlgorithm*

Name = 'Particle Swarm Optimization'

select_features(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
numpy.ndarray[bool]: Mask of selected features.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.preprocessing.feature_selection.**SelectKBest**(***kwargs*)

Bases: *FeatureSelectionAlgorithm*

Implementation of feature selection using selection of k best features according to used score function.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

See Also:

- *niaaml.preprocessing.feature_selection.feature_selection_algorithm.FeatureSelectionAlgorithm*

Name = 'Select K Best'

select_features(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
numpy.ndarray[bool]: Mask of selected features.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.preprocessing.feature_selection.**SelectPercentile**(***kwargs*)

Bases: *FeatureSelectionAlgorithm*

Implementation of feature selection using percentile selection of best features according to used score function.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html

See Also:

- *niaaml.preprocessing.feature_selection.feature_selection_algorithm.FeatureSelectionAlgorithm*

Name = 'Select Percentile'

select_features(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
numpy.ndarray[bool]: Mask of selected features.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.preprocessing.feature_selection.**VarianceThreshold**(***kwargs*)

Bases: *FeatureSelectionAlgorithm*

Implementation of feature selection using variance threshold.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html

See Also:

- *niaaml.preprocessing.feature_selection.feature_selection_algorithm.FeatureSelectionAlgorithm*

Name = 'Variance Threshold'

select_features(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
numpy.ndarray[bool]: Mask of selected features.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

class niaaml.preprocessing.feature_selection.**jDEFSTH**(***kwargs*)

Bases: *FeatureSelectionAlgorithm*

Implementation of self-adaptive differential evolution for feature selection using threshold mechanism.

Date:
2020

Author:
Iztok Fister Jr.

Reference:

D. Fister, I. Fister, T. Jagrič, I. Fister Jr., J. Brest. A novel self-adaptive differential evolution for feature selection using threshold mechanism . In: Proceedings of the 2018 IEEE Symposium on Computational Intelligence (SSCI 2018), pp. 17-24, 2018.

Reference URL:
http://iztok-jr-fister.eu/static/publications/236.pdf

License:
MIT

See Also:

- *niaaml.preprocessing.feature_selection.feature_selection_algorithm.FeatureSelectionAlgorithm*

`Name = 'Self-Adaptive Differential Evolution'`

`select_features`(*x*, *y*, ***kwargs*)

Perform the feature selection process.

Arguments:
x (pandas.core.frame.DataFrame): Array of original features. y (pandas.core.series.Series) Expected classifier results.

Returns:
numpy.ndarray[bool]: Mask of selected features.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

6.4.2 niaaml.preprocessing.feature_transform

`class` niaaml.preprocessing.feature_transform.`FeatureTransformAlgorithm`(***kwargs*)

Bases: *PreprocessingAlgorithm*

Class for implementing feature transform algorithms.

Date:
2020

Author:
Luka Pečnik

License:
MIT

See Also:

- *niaaml.preprocessing.preprocessing_algorithm.PreprocessingAlgorithm*

`fit`(*x*, ***kwargs*)

Fit implemented feature transform algorithm.

Arguments:
x (pandas.core.frame.DataFrame): n samples to fit transformation algorithm.

transform(*x*, ***kwargs*)

Transforms the given x data.

Arguments:
x (pandas.core.frame.DataFrame): Data to transform.

Returns:
pandas.core.frame.DataFrame: Transformed data.

class niaaml.preprocessing.feature_transform.**FeatureTransformAlgorithmFactory**(***kwargs*)

Bases: `Factory`

Class with string mappings to feature transform algorithms.

Attributes:
_entities (Dict[str, FeatureTransformAlgorithm]): Mapping from strings to feature transform algorithms.

class niaaml.preprocessing.feature_transform.**MaxAbsScaler**(***kwargs*)

Bases: *FeatureTransformAlgorithm*

Implementation of feature scaling by its maximum absolute value.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html#sklearn.preprocessing.MaxAbsScaler

See Also:

- *niaaml.preprocessing.feature_transform.FeatureTransformAlgorithm*

Name = 'Maximum Absolute Scaler'

fit(*x*, ***kwargs*)

Fit implemented transformation algorithm.

Arguments:
x (pandas.core.frame.DataFrame): n samples to fit transformation algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

transform(*x*, ***kwargs*)

Transforms the given x data.

Arguments:
x (pandas.core.frame.DataFrame): Data to transform.

Returns:
pandas.core.frame.DataFrame: Transformed data.

class niaaml.preprocessing.feature_transform.**Normalizer**(***kwargs*)

Bases: *FeatureTransformAlgorithm*

Implementation of feature normalization algorithm.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer

See Also:

- *niaaml.preprocessing.feature_transform.FeatureTransformAlgorithm*

Name = 'Normalizer'

fit(*x*, ***kwargs*)

Fit implemented transformation algorithm.

Arguments:
x (pandas.core.frame.DataFrame): n samples to fit transformation algorithm.

set_parameters(***kwargs*)

Set the parameters/arguments of the algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

transform(*x*, ***kwargs*)

Transforms the given x data.

Arguments:
x (pandas.core.frame.DataFrame): Data to transform.

Returns:
pandas.core.frame.DataFrame: Transformed data.

class niaaml.preprocessing.feature_transform.**QuantileTransformer**(***kwargs*)

Bases: *FeatureTransformAlgorithm*

Implementation of quantile transformer.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html#sklearn.preprocessing.QuantileTransformer

See Also:

- *niaaml.preprocessing.feature_transform.FeatureTransformAlgorithm*

Name = 'Quantile Transformer'

fit(*x*, ***kwargs*)

Fit implemented transformation algorithm.

Arguments:
x (pandas.core.frame.DataFrame): n samples to fit transformation algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

transform(*x*, ***kwargs*)

Transforms the given x data.

Arguments:
x (pandas.core.frame.DataFrame): Data to transform.

Returns:
pandas.core.frame.DataFrame: Transformed data.

class niaaml.preprocessing.feature_transform.**RobustScaler**(***kwargs*)

Bases: *FeatureTransformAlgorithm*

Implementation of the robust scaler.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html#sklearn.preprocessing.RobustScaler

See Also:

- *niaaml.preprocessing.feature_transform.FeatureTransformAlgorithm*

Name = 'Robust Scaler'

fit(*x*, ***kwargs*)

Fit implemented transformation algorithm.

Arguments:
x (pandas.core.frame.DataFrame): n samples to fit transformation algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

transform(*x*, ***kwargs*)

Transforms the given x data.

Arguments:
x (pandas.core.frame.DataFrame): Data to transform.

Returns:
pandas.core.frame.DataFrame: Transformed data.

class niaaml.preprocessing.feature_transform.**StandardScaler**(***kwargs*)

Bases: *FeatureTransformAlgorithm*

Implementation of feature standard scaling algorithm.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

See Also:

- *niaaml.preprocessing.feature_transform.FeatureTransformAlgorithm*

Name = 'Standard Scaler'

fit(*x*, ***kwargs*)

Fit implemented transformation algorithm.

Arguments:
x (pandas.core.frame.DataFrame): n samples to fit transformation algorithm.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

transform(*x*, ***kwargs*)

Transforms the given x data.

Arguments:
x (pandas.core.frame.DataFrame): Data to transform.

Returns:
pandas.core.frame.DataFrame: Transformed data.

6.4.3 niaaml.preprocessing.encoding

class niaaml.preprocessing.encoding.**EncoderFactory**(***kwargs*)

Bases: Factory

Class with string mappings to encoders.

Attributes:
_entities (Dict[str, FeatureEncoder]): Mapping from strings to encoders.

See Also:

- niaaml.utilities.Factory

class niaaml.preprocessing.encoding.**FeatureEncoder**(***kwargs*)

Bases: object

Class for implementing feature encoders.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
Name (str): Name of the feature encoder.

Name = None

fit(*feature*)

Fit feature encoder.

Arguments:
feature (pandas.core.frame.DataFrame): A column (categorical) from DataFrame of features.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

transform(*feature*)

Transform feature's values.

Arguments:
feature (pandas.core.frame.DataFrame): A column (categorical) from DataFrame of features.

Returns:
pandas.core.frame.DataFrame: A transformed column.

class niaaml.preprocessing.encoding.**OneHotEncoder**(***kwargs*)

Bases: *FeatureEncoder*

Implementation of one-hot encoder.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Reference:
Seger, Cedric. "An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing." (2018).

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

See Also:

- *niaaml.preprocessing.encoding.FeatureEncoder*

`Name = 'One-Hot Encoder'`

`fit`(*feature*)

Fit feature encoder.

Arguments:
feature (pandas.core.frame.DataFrame): A column (categorical) from DataFrame of features.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

`transform`(*feature*)

Transform feature's values.

Arguments:
feature (pandas.core.frame.DataFrame): A column (categorical) from DataFrame of features.

Returns:
pandas.core.frame.DataFrame: A transformed column.

niaaml.preprocessing.encoding.**encode_categorical_features**(*features*, *encoder*)

Encode categorical features.

Arguments:
features (pandas.core.frame.DataFrame): DataFrame of features. encoder (str): Name of the encoder to use.

Returns:

Tuple[pandas.core.frame.DataFrame, Iterable[FeatureEncoder]]:

1. Converted dataframe.
2. Dictionary of encoders for all categorical features.

6.4.4 niaaml.preprocessing.imputation

class niaaml.preprocessing.imputation.**Imputer**(***kwargs*)

Bases: object

Class for implementing imputers.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
Name (str): Name of the imputer.

Name = None

fit(*feature*)

Fit imputer.

Arguments:
feature (pandas.core.frame.DataFrame): A column from DataFrame of features.

to_string()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

transform(*feature*)

Transform feature's values.

Arguments:
feature (pandas.core.frame.DataFrame): A column from DataFrame of features.

Returns:
pandas.core.frame.DataFrame: A transformed column.

class niaaml.preprocessing.imputation.**ImputerFactory**(***kwargs*)

Bases: Factory

Class with string mappings to imputers.

Attributes:
_entities (Dict[str, Imputer]): Mapping from strings to imputers.

See Also:

- niaaml.utilities.Factory

class niaaml.preprocessing.imputation.**SimpleImputer**(***kwargs*)

Bases: *Imputer*

Implementation of simple imputer.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

See Also:

- *niaaml.preprocessing.imputation.Imputer*

`Name = 'Simple Imputer'`

`fit`(*feature*)

Fit imputer.

Arguments:
feature (pandas.core.frame.DataFrame): A column from DataFrame of features.

`to_string`()

User friendly representation of the object.

Returns:
str: User friendly representation of the object.

`transform`(*feature*)

Transform feature's values.

Arguments:
feature (pandas.core.frame.DataFrame): A column from DataFrame of features.

Returns:
pandas.core.frame.DataFrame: A transformed column.

niaaml.preprocessing.imputation.**impute_features**(*features*, *imputer*)

Impute features with missing data.

Arguments:
features (pandas.core.frame.DataFrame): DataFrame of features. imputer (str): Name of the imputer to use.

Returns:

Tuple[pandas.core.frame.DataFrame, Dict[Imputer]]:

1. Converted dataframe.
2. Dictionary of imputers for all features with missing data.

6.5 niaaml.fitness

class niaaml.fitness.**Accuracy**(***kwargs*)

Bases: *FitnessFunction*

Class representing the accuracy as a fitness function.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

See Also:

- *niaaml.fitness.FitnessFunction*

`Name = 'Accuracy'`

`get_fitness`(*predicted*, *expected*)

Return fitness value. The larger return value should represent a better fitness for the framework to work properly.

Arguments:
predicted (pandas.core.series.Series): Predicted values. expected (pandas.core.series.Series): Expected values.

Returns:
float: Calculated fitness value.

`class niaaml.fitness.CohenKappa`(***kwargs*)

Bases: *FitnessFunction*

Class representing the cohen's kappa as a fitness function.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html

See Also:

- *niaaml.fitness.FitnessFunction*

`Name = "Cohen's Kappa"`

`get_fitness`(*predicted*, *expected*)

Return fitness value. The larger return value should represent a better fitness for the framework to work properly.

Arguments:
predicted (pandas.core.series.Series): Predicted values. expected (pandas.core.series.Series): Expected values.

Returns:
float: Calculated fitness value.

`class niaaml.fitness.F1`(***kwargs*)

Bases: *FitnessFunction*

Class representing the F1-score as a fitness function.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

See Also:

- *niaaml.fitness.FitnessFunction*

Name = 'F-score'

get_fitness(*predicted*, *expected*)

Return fitness value. The larger return value should represent a better fitness for the framework to work properly.

Arguments:
predicted (pandas.core.series.Series): Predicted values. expected (pandas.core.series.Series): Expected values.

Returns:
float: Calculated fitness value.

class niaaml.fitness.**FitnessFactory**(***kwargs*)

Bases: Factory

Class with string mappings to fitness class.

Attributes:
_entities (Dict[str, Fitness]): Mapping from strings to fitness classes.

See Also:

- niaaml.utilities.Factory

class niaaml.fitness.**FitnessFunction**(***kwargs*)

Bases: object

Class for implementing fitness functions.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Attributes:
Name (str): Name of the fitness function.

Name = None

get_fitness(*predicted*, *expected*)

Return fitness value. The larger return value should represent a better fitness for the framework to work properly.

Arguments:
predicted (pandas.core.series.Series): Predicted values. expected (pandas.core.series.Series): Expected values.

Returns:
float: Calculated fitness value.

`set_parameters`(***kwargs*)

Set the parameters/arguments of the pipeline component.

`class niaaml.fitness.Precision`(***kwargs*)

Bases: *FitnessFunction*

Class representing the precision as a fitness function.

Date:
2020

Author:
Luka Pečnik

License:
MIT

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

See Also:

- *niaaml.fitness.FitnessFunction*

`Name = 'Precision'`

`get_fitness`(*predicted*, *expected*)

Return fitness value. The larger return value should represent a better fitness for the framework to work properly.

Arguments:
predicted (pandas.core.series.Series): Predicted values. expected (pandas.core.series.Series): Expected values.

Returns:
float: Calculated fitness value.

CHAPTER

SEVEN

ABOUT

NiaAML is an automated machine learning Python framework based on nature-inspired algorithms for optimization. The name comes from the automated machine learning method of the same name. Its goal is to efficiently compose the best possible classification pipeline for the given task using components on the input. The components are divided into three groups: feature seletion algorithms, feature transformation algorithms and classifiers. The framework uses nature-inspired algorithms for optimization to choose the best set of components for the classification pipeline on the output and optimize their parameters. We use NiaPy framework for the optimization process which is a popular Python collection of nature-inspired algorithms. The NiaAML framework is easy to use and customize or expand to suit your needs.

The NiaAML framework allows you not only to run full pipeline optimization, but also separate implemented components such as classifiers, feature selection algorithms, etc. It supports numerical and categorical features.

7.1 Licence

This package is distributed under the MIT License.

7.2 Disclaimer

This framework is provided as-is, and there are no guarantees that it fits your purposes or that it is bug-free. Use it at your own risk!

CHAPTER

EIGHT

CONTRIBUTING TO NIAAML

First off, thanks for taking the time to contribute!

8.1 Code of Conduct

This project and everyone participating in it is governed by the *Contributor Covenant Code of Conduct*. By participating, you are expected to uphold this code. Please report unacceptable behavior to lukapecnik96@gmail.com.

8.2 How Can I Contribute?

8.2.1 Reporting Bugs

Before creating bug reports, please check existing issues list as you might find out that you don't need to create one. When you are creating a bug report, please include as many details as possible in the issue template.

8.2.2 Suggesting Enhancements

Open new issue using the feature request template.

8.2.3 Pull requests

Fill in the pull request template and make sure your code is documented.

CHAPTER

NINE

CONTRIBUTOR COVENANT CODE OF CONDUCT

9.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

9.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

9.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

9.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

9.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at lukapecnik96@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

9.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

9.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

9.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

9.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

9.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

9.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

CHAPTER

TEN

REFERENCES

[1] Iztok Fister Jr., Milan Zorman, Dušan Fister, Iztok Fister. Continuous optimizers for automatic design and evaluation of classification pipelines. In: Frontier applications of nature inspired computation. Springer tracts in nature-inspired computing, pp.281-301, 2020.

PYTHON MODULE INDEX

n

INDEX

I

J

K

L

M

N

O

P

Q

R

S

T

V