ITMO$_{FS}$

Release 0.3.2

Jun 28, 2021

Getting Started

CHAPTER 1

Install and contribution

1.1 Prerequisites

The feature selection library requires the following dependencies:

- python (>=3.6)
- numpy (>=1.13.3)
- scipy (>=0.19.1)
- scikit-learn (>=0.22)
- imblearn (>=0.0)
- qpsolvers (>=1.0.1)

1.2 Install

ITMO_FS is currently available on the PyPi's repositories and you can install it via *pip*:

```
pip install -U ITMO_FS
```

If you prefer, you can clone it and run the setup.py file. Use the following commands to get a copy from Github and install all dependencies:

```
git clone https://github.com/LastShekel/ITMO_FS.git
cd ITMO_FS
pip install .
```

Or install using pip and GitHub:

```
pip install -U git+https://github.com/LastShekel/ITMO_FS.git
```

1.3 Test and coverage

You want to test the code before to install:

```
$ make test
```

You wish to test the coverage of your version:

```
$ make coverage
```

You can also use *pytest*:

```
$ pytest ITMO_FS -v
```

1.4 Contribute

You can contribute to this code through Pull Request on GitHub. Please, make sure that your code is coming with unit tests to ensure full coverage and continuous integration in the API.

CHAPTER 2

User Guide

2.1 Introduction

2.1.1 API's of feature selectors

Available selectors follow the scikit-learn API using the base estimator and selector mixin:

Transformer The base object, implements a `fit` method to learn from data, either:

```
selector.fit(data, targets)
```

To select features from a data set after learning, each selector implements:

```
data_selected = selector.transform(data)
```

To learn from data and select features from the same data set at once, each selector implements:

```
data_selected = selector.fir_transform(data, targets)
```

To reverse the selection operation, each selector implements:

```
data_reversed = selector.fir_transform(data)
```

Feature selectors accept the same inputs that in scikit-learn:

- `data`: array-like (2-D list, pandas.Dataframe, numpy.array) or sparse matrices;
- `targets`: array-like (1-D list, pandas.Series, numpy.array).

The output will be of the following type:

- **`data_selected`: array-like (2-D list, pandas.Dataframe, numpy.array) or** sparse matrices;
- **`data_reversed`: array-like (2-D list, pandas.Dataframe, numpy.array) or** sparse matrices;

Sparse input

For sparse input the data is **converted to the Compressed Sparse Rows representation** (see `scipy.sparse.csr_matrix`) before being fed to the sampler. To avoid unnecessary memory copies, it is recommended to choose the CSR representation upstream.

2.1.2 Problem statement regarding data sets with redundant features

Feature selection methods can be used to identify and remove unneeded, irrelevant and redundant attributes from data that do not contribute to the accuracy of a predictive model or may in fact decrease the accuracy of the model. Fewer attributes is desirable because it reduces the complexity of the model, and a simpler model is simpler to understand and explain.

Here is one of examples of feature selection improving the classification quality:

```
>>> from sklearn.datasets import make_classification
>>> from sklearn.linear_model import SGDClassifier
>>> from ITMO_FS.embedded import MOS

>>> X, y = make_classification(n_samples=300, n_features=10, random_state=0, n_
↪informative=2)
>>> sel = MOS()
>>> trX = sel.fit_transform(X, y, smote=False)

>>> cl1 = SGDClassifier()
>>> cl1.fit(X, y)
>>> cl1.score(X, y)
0.9033333333333333

>>> cl2 = SGDClassifier()
>>> cl2.fit(trX, y)
>>> cl2.score(trX, y)
0.9433333333333334
```

As expected, the quality of the SVGClassifier's results is impacted by the presence of redundant features in data set. We can see that after using of feature selection the mean accuracy increases from 0.903 to 0.943.

CHAPTER 3

ITMO_FS API

This is the full API documentation of the *ITMO_FS* toolbox.

3.1 `ITMO_FS.filters`: Filter methods

3.1.1 `ITMO_FS.filters.univariate`: Univariate filter methods

filters.univariate.VDM([weighted, q])	Creates Value Difference Metric builder.
filters.univariate.UnivariateFilter(measure)	Basic interface for using univariate measures for feature selection.

ITMO_FS.filters.univariate.VDM

class ITMO_FS.filters.univariate.**VDM**(*weighted=True*, *q=1*)

Creates Value Difference Metric builder. For continious features discretesation requered.

Parameters

- **weighted** (*bool*) – If weighted = False, modified version of metric which omits the weights is used
- **q** (*int*) – Power in VDM usually 1 or 2

Notes

For more details see papers about Improved Heterogeneous Distance Functions and Implicit Future Selection with the VDM.

Examples

```
>>> x = np.array([[0, 0, 0, 0],
...               [1, 0, 1, 1],
...               [1, 0, 0, 2]])
>>> y = np.array([0,
...               1,
...               1])
>>> vdm = VDM()
>>> vdm.fit(x, y)
array([[0.         4.35355339 4.        ]
       [4.5        0.         0.5       ]
       [4.         0.35355339 0.        ]])
```

__init__ (*weighted=True*, *q=1*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)

Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.filters.univariate.UnivariateFilter

class ITMO_FS.filters.univariate.**UnivariateFilter**(*measure*, *cutting_rule=('Best by percentage', 1.0)*)

Basic interface for using univariate measures for feature selection. List of available measures is in ITMO_FS.filters.univariate.measures, also you can provide your own measure but it should suit the argument scheme for measures, i.e. take two arguments x,y and return scores for all the features in dataset x. Same applies to cutting rules.

Parameters

- **measure** (*string or callable*) – A metric name defined in GLOB_MEASURE or a callable with signature measure (sample dataset, labels of dataset samples) which should return a list of metric values for each feature in the dataset.
- **cutting_rule** (*string or callables*) – A cutting rule name defined in GLOB_CR or a callable with signature cutting_rule (features) which should return a list of features ranked by some rule.

Examples

```
>>> import numpy as np
>>> from ITMO_FS.filters.univariate import select_k_best
>>> from ITMO_FS.filters.univariate import UnivariateFilter
>>> from ITMO_FS.filters.univariate import f_ratio_measure
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> filter = UnivariateFilter(f_ratio_measure,
... select_k_best(2)).fit(x, y)
>>> filter.selected_features_
array([4, 2], dtype=int64)
>>> filter.feature_scores_
array([0.6 , 0.2 , 1.  , 0.12, 5.4 ])
```

__init__(*measure*, *cutting_rule=('Best by percentage', 1.0)*)

Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform (*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

Measures for univariate filters

filters.univariate.fit_criterion_measure(x, y)	Calculate the FitCriterion score for features.
filters.univariate.f_ratio_measure(x, y)	Calculate Fisher score for features.
filters.univariate.gini_index(x, y)	Calculate Gini index for features.
filters.univariate.su_measure(x, y)	SU is a correlation measure between the features and the class calculated via formula SU(X,Y) = 2 * I(X|Y) / (H(X) + H(Y)).
filters.univariate.spearman_corr(x, y)	Calculate Spearman's correlation for each feature.
filters.univariate.pearson_corr(x, y)	Calculate Pearson's correlation for each feature.
filters.univariate.fechner_corr(x, y)	Calculate Sample sign correlation (Fechner correlation) for each feature.
filters.univariate.kendall_corr(x, y)	Calculate Sample sign correlation (Kendall correlation) for each feature.
filters.univariate.reliefF_measure(x, y[, ...])	Calculate ReliefF measure for each feature.
filters.univariate.chi2_measure(x, y)	Calculate the Chi-squared measure for each feature.
filters.univariate.information_gain(x, y)	Calculate mutual information for each feature by formula I(X,Y) = H(Y) - H(Y|X).

ITMO_FS.filters.univariate.fit_criterion_measure

ITMO_FS.filters.univariate.**fit_criterion_measure**(*x*, *y*)

Calculate the FitCriterion score for features. Bigger values mean more important features.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns **array-like, shape (n_features,)**

Return type feature scores

See also:

https() //core.ac.uk/download/pdf/191234514.pdf

Examples

```
>>> from ITMO_FS.filters.univariate import fit_criterion_measure
>>> import numpy as np
>>> x = np.array([[1, 2, 4, 1, 1], [2, 2, 2, 1, 2], [3, 5, 1, 1, 4],
... [1, 1, 1, 1, 4], [2, 2, 2, 1, 5]])
>>> y = np.array([1, 2, 3, 1, 2])
>>> fit_criterion_measure(x, y)
array([1. , 0.8, 0.8, 0.4, 0.6])
```

ITMO_FS.filters.univariate.f_ratio_measure

ITMO_FS.filters.univariate.**f_ratio_measure**(*x*, *y*)

Calculate Fisher score for features. Bigger values mean more important features.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns **array-like, shape (n_features,)**

Return type feature scores

See also:

https() //papers.nips.cc/paper/2909-laplacian-score-for-feature-selection.pdf

Examples

```
>>> from ITMO_FS.filters.univariate import f_ratio_measure
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> f_ratio_measure(x, y)
array([0.6 , 0.2 , 1.  , 0.12, 5.4 ])
```

ITMO_FS.filters.univariate.gini_index

ITMO_FS.filters.univariate.**gini_index**(*x*, *y*)

Calculate Gini index for features. Bigger values mean more important features. This measure works best with discrete features due to being based on information theory.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns **array-like, shape (n_features,)**

Return type feature scores

See also:

http() //lkm.fri.uni-lj.si/xaigor/slo/clanki/ijcai95z.pdf

Examples

```
>>> from ITMO_FS.filters.univariate import gini_index
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> gini_index(x, y)
array([0.14      , 0.04      , 0.64      , 0.24      , 0.37333333])
```

ITMO_FS.filters.univariate.su_measure

ITMO_FS.filters.univariate.**su_measure**(*x*, *y*)

SU is a correlation measure between the features and the class calculated via formula SU(X,Y) = 2 * I(X|Y) / (H(X) + H(Y)). Bigger values mean more important features. This measure works best with discrete features due to being based on information theory.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns **array-like, shape (n_features,)**

Return type feature scores

See also:

https() //pdfs.semanticscholar.org/9964/c7b42e6ab311f88e493b3fc552515e0c764a.pdf

Examples

```
>>> from ITMO_FS.filters.univariate import su_measure
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> su_measure(x, y)
array([0.28694182, 0.13715115, 0.79187567, 0.47435099, 0.67126949])
```

ITMO_FS.filters.univariate.spearman_corr

ITMO_FS.filters.univariate.**spearman_corr**(*x*, *y*)

Calculate Spearman's correlation for each feature. Bigger absolute values mean more important features. This measure works best with discrete features due to being based on statistics.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.

- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns array-like, shape (n_features,)

Return type feature scores

See also:

https() //en.wikipedia.org/wiki/Spearman's_rank_correlation_coefficient

Examples

```
>>> from ITMO_FS.filters.univariate import spearman_corr
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> spearman_corr(x, y)
array([-0.186339  ,  0.30429031, -0.52704628, -0.30555556,  0.35355339])
```

ITMO_FS.filters.univariate.pearson_corr

ITMO_FS.filters.univariate.**pearson_corr**(*x*, *y*)

Calculate Pearson's correlation for each feature. Bigger absolute values mean more important features. This measure works best with discrete features due to being based on statistics.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns array-like, shape (n_features,)

Return type feature scores

See also:

https() //en.wikipedia.org/wiki/Pearson_correlation_coefficient

Examples

```
>>> from ITMO_FS.filters.univariate import pearson_corr
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> pearson_corr(x, y)
array([-0.13363062,  0.32732684, -0.60631301, -0.26244533,  0.53452248])
```

ITMO_FS.filters.univariate.fechner_corr

ITMO_FS.filters.univariate.**fechner_corr**(*x*, *y*)

Calculate Sample sign correlation (Fechner correlation) for each feature. Bigger absolute values mean more important features.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns **array-like, shape (n_features,)**

Return type feature scores

Examples

```
>>> from ITMO_FS.filters.univariate import fechner_corr
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> fechner_corr(x, y)
array([-0.2,  0.2, -0.4, -0.2, -0.2])
```

ITMO_FS.filters.univariate.kendall_corr

ITMO_FS.filters.univariate.**kendall_corr**(*x*, *y*)

Calculate Sample sign correlation (Kendall correlation) for each feature. Bigger absolute values mean more important features.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns **array-like, shape (n_features,)**

Return type feature scores

See also:

https() //en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient

Examples

```
>>> from ITMO_FS.filters.univariate import kendall_corr
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
```

(continues on next page)

(continued from previous page)

```
>>> kendall_corr(x, y)
array([-0.1,  0.2, -0.4, -0.2,  0.2])
```

ITMO_FS.filters.univariate.reliefF_measure

ITMO_FS.filters.univariate.**reliefF_measure**(*x*, *y*, *k_neighbors=1*)

Calculate ReliefF measure for each feature. Bigger values mean more important features.

Note: Only for complete x Rather than repeating the algorithm m(TODO Ask Nikita about user defined) times, implement it exhaustively (i.e. n times, once for each instance) for relatively small n (up to one thousand).

Calculates spearman correlation for each feature. Spearman's correlation assesses monotonic relationships (whether linear or not). If there are no repeated data values, a perfect Spearman correlation of +1 or 1 occurs when each of the variables is a perfect monotone function of the other.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The input samples.
- **y** (*array-like, shape (n_samples,)*) – The classes for the samples.
- **k_neighbors** (*int, optional*) – The number of neighbors to consider when assigning feature importance scores. More neighbors results in more accurate scores but takes longer. Selection of k hits and misses is the basic difference to Relief and ensures greater robustness of the algorithm concerning noise.

Returns **array-like, shape (n_features,)**

Return type feature scores

See also:

R.J.(),review.()

Examples

```
>>> from ITMO_FS.filters.univariate import reliefF_measure
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1], [1, 2, 1, 4, 2], [4, 3, 2, 3, 1]])
>>> y = np.array([1, 2, 2, 1, 2, 1, 2])
>>> reliefF_measure(x, y)
array([-0.14285714, -0.57142857,  0.10714286, -0.14285714,  0.07142857])
>>> reliefF_measure(x, y, k_neighbors=2)
array([-0.07142857, -0.17857143, -0.07142857, -0.0952381 , -0.17857143])
```

ITMO_FS.filters.univariate.chi2_measure

ITMO_FS.filters.univariate.**chi2_measure**(*x*, *y*)

Calculate the Chi-squared measure for each feature. Bigger values mean more important features. This measure works best with discrete features due to being based on statistics.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.

- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns array-like, shape (n_features,)

Return type feature scores

See also:

http() //lkm.fri.uni-lj.si/xaigor/slo/clanki/ijcai95z.pdf

Example

```
>>> from ITMO_FS.filters.univariate import chi2_measure
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> chi2_measure(x, y)
array([ 1.875     ,  0.83333333, 10.        ,  3.75      ,  6.66666667])
```

ITMO_FS.filters.univariate.information_gain

ITMO_FS.filters.univariate.**information_gain**(*x*, *y*)

Calculate mutual information for each feature by formula I(X,Y) = H(Y) - H(Y|X). Bigger values mean more important features. This measure works best with discrete features due to being based on information theory.

Parameters

- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns array-like, shape (n_features,)

Return type feature scores

Examples

```
>>> from ITMO_FS.filters.univariate import information_gain
>>> import numpy as np
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> information_gain(x, y)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
```

Cutting rules for univariate filters

filters.univariate.select_best_by_value(value)	
filters.univariate.select_worst_by_value(value)	
filters.univariate.select_k_best(k)	
filters.univariate.select_k_worst(k)	
filters.univariate.select_best_percentage(percent)	
filters.univariate.select_worst_percentage(percent)	

ITMO_FS.filters.univariate.select_best_by_value

ITMO_FS.filters.univariate.**select_best_by_value**(*value*)

ITMO_FS.filters.univariate.select_worst_by_value

ITMO_FS.filters.univariate.**select_worst_by_value**(*value*)

ITMO_FS.filters.univariate.select_k_best

ITMO_FS.filters.univariate.**select_k_best**(*k*)

ITMO_FS.filters.univariate.select_k_worst

ITMO_FS.filters.univariate.**select_k_worst**(*k*)

ITMO_FS.filters.univariate.select_best_percentage

ITMO_FS.filters.univariate.**select_best_percentage**(*percent*)

ITMO_FS.filters.univariate.select_worst_percentage

ITMO_FS.filters.univariate.**select_worst_percentage**(*percent*)

3.1.2 ITMO_FS.filters.multivariate: Multivariate filter methods

filters.multivariate.DISRWithMassive(n_features)	Create DISR (Double Input Symmetric Relevance) feature selection filter based on kASSI criterin for feature selection which aims at maximizing the mutual information avoiding, meanwhile, large multivariate density estimation.

Continued on next page

Table 4 – continued from previous page

filters.multivariate.FCBFDiscreteFilter([delta])	Create FCBF (Fast Correlation Based filter) feature selection filter based on mutual information criteria for data with discrete features.
filters.multivariate.MultivariateFilter(...)	Provides basic functionality for multivariate filters.
filters.multivariate.STIR(n_features[, ...])	Feature selection using STIR algorithm.
filters.multivariate.TraceRatioFisher(n_features)	Creates TraceRatio(similarity based) feature selection filter performed in supervised way, i.e.
filters.multivariate.MIMAGA(mim_size, pop_size)	

ITMO_FS.filters.multivariate.DISRWithMassive

class ITMO_FS.filters.multivariate.**DISRWithMassive**(*n_features*)

Create DISR (Double Input Symmetric Relevance) feature selection filter based on kASSI criterin for feature selection which aims at maximizing the mutual information avoiding, meanwhile, large multivariate density estimation. Its a kASSI criterion with approximation of the information of a set of variables by counting average information of subset on combination of two features. This formulation thus deals with feature complementarity up to order two by preserving the same computational complexity of the MRMR and CMIM criteria The DISR calculation is done using graph based solution.

Parameters **n_features** (*int*) – Number of features to select.

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import DISRWithMassive
>>> import numpy as np
>>> X = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> disr = DISRWithMassive(3).fit(X, y)
>>> disr.selected_features_
array([0, 1, 4], dtype=int64)
```

__init__(*n_features*)

Initialize self. See help(type(self)) for accurate signature.

fit(*X*, *y=None*, ***fit_params*)

Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform(*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.filters.multivariate.FCBFDiscreteFilter

class ITMO_FS.filters.multivariate.**FCBFDiscreteFilter**(*delta=0.1*)
Create FCBF (Fast Correlation Based filter) feature selection filter based on mutual information criteria for data with discrete features. This filter finds best set of features by searching for a feature, which provides the most information about classification problem on given dataset at each step and then eliminating features which are less relevant than redundant.

Parameters **delta** (*float*) – Symmetric uncertainty value threshold.

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import FCBFDiscreteFilter
>>> import numpy as np
>>> X = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> fcbf = FCBFDiscreteFilter().fit(X, y)
>>> fcbf.selected_features_
array([4], dtype=int64)
```

__init__ (*delta=0.1*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform (*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.filters.multivariate.MultivariateFilter

class ITMO_FS.filters.multivariate.**MultivariateFilter** (*measure*, *n_features*, *beta=None*, *gamma=None*)
Provides basic functionality for multivariate filters.

Parameters

- **measure** (*string or callable*) – A metric name defined in GLOB_MEASURE or a callable with signature measure(selected_features, free_features, dataset, labels) which should return a list of metric values for each feature in the dataset.
- **n_features** (*int*) – Number of features to select.
- **beta** (*float, optional*) – Initialize only in case you run MIFS or generalizedCriteria metrics.
- **gamma** (*float, optional*) – Initialize only in case you run generalizedCriteria metric.

Examples

```
>>> from ITMO_FS.filters.multivariate import MultivariateFilter
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> data = est.fit_transform(x)
>>> model = MultivariateFilter('JMI', 3).fit(x, y)
>>> model.selected_features_
array([4, 0, 1], dtype=int64)
```

__init__ (*measure*, *n_features*, *beta=None*, *gamma=None*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform (*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.filters.multivariate.STIR

class ITMO_FS.filters.multivariate.**STIR**(*n_features*, *metric='manhattan'*, *k=1*)
Feature selection using STIR algorithm.

Parameters

- **n_features** (*int*) – Number of features to select.
- **metric** (*str or callable*) – Distance metric to use in kNN. If str, should be one of the standard distance metrics (e.g. 'euclidean' or 'manhattan'). If callable, should have the signature metric(x1 (array-like, shape (n,)), x2 (array-like, shape (n,))) that should return the distance between two vectors.
- **k** (*int*) – Number of constant nearest hits/misses.

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import STIR
>>> import numpy as np
>>> X = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 2, 2, 1, 2])
>>> model = STIR(2).fit(X, y)
>>> model.selected_features_
array([2, 0], dtype=int64)
```

__init__(*n_features*, *metric='manhattan'*, *k=1*)
Initialize self. See help(type(self)) for accurate signature.

fit(*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform(*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –

- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.filters.multivariate.TraceRatioFisher

class ITMO_FS.filters.multivariate.**TraceRatioFisher**(*n_features*, *epsilon=0.001*)
Creates TraceRatio(similarity based) feature selection filter performed in supervised way, i.e. fisher version

Parameters

- **n_features** (*int*) – Number of features to select.
- **epsilon** (*float*) – Lambda change threshold.

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import TraceRatioFisher
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 1, 1, 2])
>>> tracer = TraceRatioFisher(3).fit(x, y)
>>> tracer.selected_features_
array([0, 1, 3], dtype=int64)
```

__init__ (*n_features*, *epsilon=0.001*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)

Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.filters.multivariate.MIMAGA

class ITMO_FS.filters.multivariate.**MIMAGA**(*mim_size*, *pop_size*, *max_iter=20*, *f_target=0.8*, *k1=0.6*, *k2=0.3*, *k3=0.9*, *k4=0.001*)

__init__(*mim_size*, *pop_size*, *max_iter=20*, *f_target=0.8*, *k1=0.6*, *k2=0.3*, *k3=0.9*, *k4=0.001*)

Parameters

- **mim_size** – desirable number of filtered features after MIM
- **pop_size** – initial population size
- **max_iter** – maximum number of iterations in algorithm
- **f_target** – desirable fitness value
- **k1** – consts to determine crossover probability
- **k2** – consts to determine crossover probability
- **k3** – consts to determine mutation probability
- **k4** – consts to determine mutation probability

See also:

https() //www.sciencedirect.com/science/article/abs/pii/S0925231217304150

fit(*X*, *y=None*, ***fit_params*)

Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform(*X*, *y=None*, ***fit_params*)

Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

mimaga_filter(*genes*, *classes*)
The main function to run algorithm :param genes: initial dataset in format: samples are rows, features are columns :param classes: distribution pf initial dataset :return: filtered with MIMAGA dataset, fitness value

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

Measures for multivariate filters

filters.multivariate.MIM(selected_features, ...)	Mutual Information Maximization feature scoring criterion.
filters.multivariate.MRMR(selected_features, ...)	Minimum-Redundancy Maximum-Relevance feature scoring criterion.
filters.multivariate.JMI(selected_features, ...)	Joint Mutual Information feature scoring criterion.
filters.multivariate.CIFE(selected_features, ...)	Conditional Infomax Feature Extraction feature scoring criterion.

Continued on next page

Table 5 – continued from previous page

filters.multivariate.MIFS(selected_features, ...)	Mutual Information Feature Selection feature scoring criterion.
filters.multivariate.CMIM(selected_features, ...)	Conditional Mutual Info Maximisation feature scoring criterion.
filters.multivariate.ICAP(selected_features, ...)	Interaction Capping feature scoring criterion.
filters.multivariate.DCSF(selected_features, ...)	Dynamic change of selected feature with the class scoring criterion.
filters.multivariate.CFR(selected_features, ...)	The criterion of CFR maximizes the correlation and minimizes the redundancy.
filters.multivariate.MRI(selected_features, ...)	Max-Relevance and Max-Independence feature scoring criteria.
filters.multivariate.IWFS(selected_features, ...)	Interaction Weight base feature scoring criteria.
filters.multivariate.generalizedCriteria(...)	This feature scoring criteria is a linear combination of all relevance, redundancy, conditional dependency Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

ITMO_FS.filters.multivariate.MIM

ITMO_FS.filters.multivariate.**MIM**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Mutual Information Maximization feature scoring criterion. This criterion focuses only on increase of relevance. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import MIM
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
```

(continues on next page)

(continued from previous page)

```
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> MIM(np.array(selected_features), np.array(other_features), x, y)
array([1.33217904, 0.67301167, 1.60943791])
```

ITMO_FS.filters.multivariate.MRMR

ITMO_FS.filters.multivariate.**MRMR**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Minimum-Redundancy Maximum-Relevance feature scoring criterion. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import MRMR
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> MRMR(np.array(selected_features), np.array(other_features), x, y)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
```

(continues on next page)

(continued from previous page)

```
>>> MRMR(np.array(selected_features), np.array(other_features), x, y)
array([0.80471896, 0.33650583, 0.94334839])
```

ITMO_FS.filters.multivariate.JMI

ITMO_FS.filters.multivariate.**JMI**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Joint Mutual Information feature scoring criterion. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import JMI
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> JMI(np.array(selected_features), np.array(other_features), x, y)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> JMI(np.array(selected_features), np.array(other_features), x, y)
array([0.80471896, 0.33650583, 0.94334839])
```

ITMO_FS.filters.multivariate.CIFE

ITMO_FS.filters.multivariate.**CIFE**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Conditional Infomax Feature Extraction feature scoring criterion. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import CIFE
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> CIFE(np.array(selected_features), np.array(other_features), x, y)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> CIFE(np.array(selected_features), np.array(other_features), x, y)
array([0.27725887, 0.        , 0.27725887])
```

ITMO_FS.filters.multivariate.MIFS

ITMO_FS.filters.multivariate.**MIFS**(*selected_features*, *free_features*, *x*, *y*, *beta*, ***kwargs*)

Mutual Information Feature Selection feature scoring criterion. This criterion includes the I(X;Y) term to ensure feature relevance, but introduces a penalty to enforce low correlations with features already selected in set. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **beta** (*float*) – Coefficient for redundancy term.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import MIFS
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> MIFS(np.array(selected_features), np.array(other_features), x, y, 0.4)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> MIFS(np.array(selected_features), np.array(other_features), x, y, 0.4)
array([0.91021097, 0.403807  , 1.0765663 ])
```

ITMO_FS.filters.multivariate.CMIM

ITMO_FS.filters.multivariate.**CMIM**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Conditional Mutual Info Maximisation feature scoring criterion. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns array-like, shape (n_features,)

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import CMIM
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> CMIM(np.array(selected_features), np.array(other_features), x, y)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> CMIM(np.array(selected_features), np.array(other_features), x, y)
array([0.27725887, 0.        , 0.27725887])
```

ITMO_FS.filters.multivariate.ICAP

ITMO_FS.filters.multivariate.**ICAP** (*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Interaction Capping feature scoring criterion. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns array-like, shape (n_features,)

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import ICAP
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> ICAP(np.array(selected_features), np.array(other_features), x, y)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> ICAP(np.array(selected_features), np.array(other_features), x, y)
array([0.27725887, 0.        , 0.27725887])
```

ITMO_FS.filters.multivariate.DCSF

ITMO_FS.filters.multivariate.**DCSF**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Dynamic change of selected feature with the class scoring criterion. DCSF employs both mutual information and conditional mutual information to find an optimal subset of features. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import DCSF
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
```

(continues on next page)

(continued from previous page)

```
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> DCSF(np.array(selected_features), np.array(other_features), x, y)
array([0., 0., 0., 0., 0.])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> DCSF(np.array(selected_features), np.array(other_features), x, y)
array([0.83177662, 0.65916737, 0.55451774])
```

ITMO_FS.filters.multivariate.CFR

ITMO_FS.filters.multivariate.**CFR**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

The criterion of CFR maximizes the correlation and minimizes the redundancy. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import CFR
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
```

(continues on next page)

(continued from previous page)

```
>>> CFR(np.array(selected_features), np.array(other_features), x, y)
array([0., 0., 0., 0., 0.])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> CFR(np.array(selected_features), np.array(other_features), x, y)
array([0.55451774, 0.        , 0.55451774])
```

ITMO_FS.filters.multivariate.MRI

ITMO_FS.filters.multivariate.**MRI**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Max-Relevance and Max-Independence feature scoring criteria. Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import MRI
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> MRI(np.array(selected_features), np.array(other_features), x, y)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> MRI(np.array(selected_features), np.array(other_features), x, y)
array([0.62889893, 0.22433722, 0.72131855])
```

ITMO_FS.filters.multivariate.IWFS

ITMO_FS.filters.multivariate.**IWFS**(*selected_features*, *free_features*, *x*, *y*, ***kwargs*)

Interaction Weight base feature scoring criteria. IWFS is good at identifyng Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **kwargs** (*dict, optional*) – Additional parameters to pass to generalizedCriteria.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.multivariate import IWFS
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> IWFS(np.array(selected_features), np.array(other_features), x, y)
array([0., 0., 0., 0., 0.])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> IWFS(np.array(selected_features), np.array(other_features), x, y)
array([1.0824043 , 1.11033338, 1.04268505])
```

ITMO_FS.filters.multivariate.generalizedCriteria

ITMO_FS.filters.multivariate.**generalizedCriteria**(*selected_features*, *free_features*, *x*, *y*, *beta*, *gamma*, ***kwargs*)

This feature scoring criteria is a linear combination of all relevance, redundancy, conditional dependency Given set of already selected features and set of remaining features on dataset X with labels y selects next feature.

Parameters

- **selected_features** (*list of ints*) – already selected features
- **free_features** (*list of ints*) – free features
- **x** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.
- **beta** (*float*) – Coefficient for redundancy term.
- **gamma** (*float*) – Coefficient for conditional dependancy term.

Returns **array-like, shape (n_features,)**

Return type feature scores

Notes

See the original paper[1] for more details.

References

Likelihood Maximisation: A Unifying Framework for Information Theoretic Feature Selection." JMLR 2012.

Examples

```
>>> from ITMO_FS.filters.multivariate import CFR
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> import numpy as np
>>> est = KBinsDiscretizer(n_bins=10, encode='ordinal')
>>> x = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [3, 1, 3, 1, 4], [4, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 3, 4, 5])
>>> x = est.fit_transform(x)
>>> selected_features = []
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> generalizedCriteria(np.array(selected_features),
... np.array(other_features), x, y, 0.4, 0.3)
array([1.33217904, 1.33217904, 0.        , 0.67301167, 1.60943791])
>>> selected_features = [1, 2]
>>> other_features = [i for i in range(0, x.shape[1]) if i
... not in selected_features]
>>> generalizedCriteria(np.array(selected_features),
... np.array(other_features), x, y, 0.4, 0.3)
array([0.91021097, 0.403807  , 1.0765663 ])
```

3.1.3 ITMO_FS.filters.unsupervised: Unsupervised filter methods

filters.unsupervised.TraceRatioLaplacian(...)	TraceRatio(similarity based) feature selection filter performed in unsupervised way, i.e laplacian version

[1] Brown, Gavin et al. "Conditional

ITMO_FS.filters.unsupervised.TraceRatioLaplacian

class ITMO_FS.filters.unsupervised.**TraceRatioLaplacian**(*n_features*, *k=5*, *t=1*, *epsilon=0.001*)

TraceRatio(similarity based) feature selection filter performed in unsupervised way, i.e laplacian version

Parameters

- **n_features** (*int*) – Amount of features to select.
- **k** (*int*) – Amount of nearest neighbors to use while building the graph.
- **t** (*int*) – constant for kernel function calculation
- **epsilon** (*float*) – Lambda change threshold.

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.filters.unsupervised import TraceRatioLaplacian
>>> import numpy as np
>>> X = np.array([[1, 2, 3, 3, 1], [2, 2, 3, 3, 2], [1, 3, 3, 1, 3],
... [1, 1, 3, 1, 4], [2, 4, 3, 1, 5]])
>>> y = np.array([1, 2, 1, 1, 2])
>>> tracer = TraceRatioLaplacian(2, k=2).fit(X)
>>> tracer.selected_features_
array([3, 1], dtype=int64)
```

__init__(*n_features*, *k=5*, *t=1*, *epsilon=0.001*)

Initialize self. See help(type(self)) for accurate signature.

fit(*X*, *y=None*, ***fit_params*)

Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform(*X*, *y=None*, ***fit_params*)

Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.

- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

3.1.4 ITMO_FS.filters.sparse: Sparse filter methods

filters.sparse.MCFS
filters.sparse.NDFS
filters.sparse.RFS
filters.sparse.SPEC
filters.sparse.UDFS

3.2 ITMO_FS.ensembles: Ensemble methods

3.2.1 ITMO_FS.ensembles.measure_based: Measure based ensemble methods

ensembles.measure_based.WeightBased(filters)	Weight-based filter ensemble.

ITMO_FS.ensembles.measure_based.WeightBased

class ITMO_FS.ensembles.measure_based.**WeightBased**(*filters*, *cutting_rule=('K best', 2)*, *fusion_function=<function weight_fusion>*, *weights=None*)

Weight-based filter ensemble. The ensemble first computes all filter scores for the dataset and then aggregates them using a selected fusion function.

Parameters

- **filters** (*collection*) – Collection of filter objects. Filters should have a fit(X, y) method and a **feature_scores_** field that contains scores for all features.
- **cutting_rule** (*string or callable*) – A cutting rule name defined in GLOB_CR or a callable with signature cutting_rule (features), which should return a list features ranked by some rule.
- **fusion_function** (*callable*) – A function with signature (filter_scores (array-like, shape (n_filters, n_features)), weights (array-like, shape (n_filters,))) that should return the aggregated weights for all features.
- **weights** (*array-like*) – An array of shape (n_filters,) defining the weights for input filters.

Examples

```
>>> from ITMO_FS.ensembles import WeightBased
>>> from ITMO_FS.filters.univariate import UnivariateFilter
>>> import numpy as np
>>> filters = [UnivariateFilter('GiniIndex'),
... UnivariateFilter('FechnerCorr'),
... UnivariateFilter('SpearmanCorr'),
... UnivariateFilter('PearsonCorr')]
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> wb = WeightBased(filters, ("K best", 2)).fit(x, y)
>>> wb.selected_features_
array([4, 1], dtype=int64)
```

__init__(*filters*, *cutting_rule=('K best', 2)*, *fusion_function=<function weight_fusion>*, *weights=None*)

Initialize self. See help(type(self)) for accurate signature.

fit(*X*, *y=None*, ***fit_params*)

Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform(*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

get_scores(*X*, *y*)
Return the normalized feature scores for all filters.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,)*) – The target values.

Returns **array-like, shape (n_filters, n_features)**

Return type feature scores

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X**(*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

3.2.2 ITMO_FS.ensembles.model_based: Model based ensemble methods

ensembles.model_based.BestSum(models, ...[, ...])	Best weighted sum ensemble.

ITMO_FS.ensembles.model_based.BestSum

class ITMO_FS.ensembles.model_based.**BestSum**(*models*, *cutting_rule*, *weight_func*, *metric='f1_micro'*, *cv=3*)

Best weighted sum ensemble. The ensemble fits the input models and computes the feature scores as the weighted sum of the models' feature scores and some performance metric (e.g. accuracy)

Parameters

- **models** (*collection*) – Collection of model objects. Models should have a fit(X, y) method and a field corresponding to feature weights.
- **cutting_rule** (*string or callable*) – A cutting rule name defined in GLOB_CR or a callable with signature cutting_rule (features), which should return a list features ranked by some rule.
- **weight_func** (*callable*) – The function to extract weights from the model.
- **metric** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable object / function with signature measure(estimator, X, y) which should return only a single value.
- **cv** (*int*) – Number of folds in cross-validation.

See also:

Jeon, H., S., Feature, 10, 3211.

Examples

```
>>> from ITMO_FS.ensembles import BestSum
>>> from sklearn.svm import SVC
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.linear_model import RidgeClassifier
>>> import numpy as np
>>> models = [SVC(kernel='linear'),
... LogisticRegression(),
... RidgeClassifier()]
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 2, 2, 1, 2])
>>> bs = BestSum(models, ("K best", 2),
... lambda model: np.square(model.coef_).sum(axis=0), cv=2).fit(x, y)
>>> bs.selected_features_
array([0, 2], dtype=int64)
```

__init__(*models*, *cutting_rule*, *weight_func*, *metric='f1_micro'*, *cv=3*)
Initialize self. See help(type(self)) for accurate signature.

fit(*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform(*X*, *y=None*, ***fit_params*)

Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)

Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)

Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

3.2.3 ITMO_FS.ensembles.ranking_based: Ranking based ensemble methods

ensembles.ranking_based.Mixed(filters, ...)	Perform feature selection based on several filters, selecting features this way: Get ranks from every filter from input.

ITMO_FS.ensembles.ranking_based.Mixed

class ITMO_FS.ensembles.ranking_based.**Mixed**(*filters*, *n_features*, *fusion_function=<function best_goes_first_fusion>*)

Perform feature selection based on several filters, selecting features this way:

Get ranks from every filter from input. Then loops through, on every iteration=i

selects features on i position on every filter then shuffles them, then adds to result list without duplication,

continues until specified number of features

Parameters

- **filters** (*collection*) – Collection of measure functions with signature measure(X, y) that should return an array of importance values for each feature.
- **n_features** (*int*) – Amount of features to select.
- **fusion_function** (*callable*) – A function with signature (filter_ranks (array-like, shape (n_filters, n_features), k (int)) that should return the indices of k selected features based on the filter rankings.

Examples

```
>>> from ITMO_FS.filters.univariate.measures import *
>>> from ITMO_FS.ensembles.ranking_based.Mixed import Mixed
>>> import numpy as np
>>> x = np.array([[3, 3, 3, 2, 2], [3, 3, 1, 2, 3], [1, 3, 5, 1, 1],
... [3, 1, 4, 3, 1], [3, 1, 2, 3, 1]])
>>> y = np.array([1, 3, 2, 1, 2])
>>> mixed = Mixed([gini_index, chi2_measure], 2).fit(x, y)
>>> mixed.selected_features_
array([2, 4], dtype=int64)
```

__init__(*filters*, *n_features*, *fusion_function=<function best_goes_first_fusion>*)
Initialize self. See help(type(self)) for accurate signature.

fit(*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform(*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

3.3 ITMO_FS.embedded: Embedded methods

embedded.MOS(model, weight_func[, loss, ...])	Perform Minimizing Overlapping Selection under SMOTE (MOSS) or under No-Sampling (MOSNS) algorithm.

3.3.1 ITMO_FS.embedded.MOS

class ITMO_FS.embedded.**MOS**(*model*, *weight_func*, *loss='log'*, *seed=42*, *l1_ratio=0.5*, *threshold=0.001*, *epochs=1000*, *alphas=array([0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19])*, *sampling=False*, *k_neighbors=2*)

Perform Minimizing Overlapping Selection under SMOTE (MOSS) or under No-Sampling (MOSNS) algorithm.

Parameters

- **model** (*object*) – The model that should have a fit(X, y) method and a field corresponding to feature weights. Currently only SGDClassifier should be passed, other models would not work.
- **weight_func** (*callable*) – The function to extract weights from the model.
- **loss** (*str, 'log' or 'hinge'*) – Loss function to use in the algorithm. 'log' gives a logistic regression, while 'hinge' gives a support vector machine.
- **seed** (*int, optional*) – Seed for python random.
- **l1_ratio** (*float*) – The value used to balance the L1 and L2 penalties in elastic-net.
- **threshold** (*float*) – The threshold value for feature dropout. Instead of comparing them to zero, they are normalized and values with absolute value lower than the threshold are dropped out.
- **epochs** (*int*) – The number of epochs to perform in the algorithm.
- **alphas** (*array-like, shape (n_alphas,), optional*) – The range of lambdas that should form the regularization path.
- **sampling** (*bool*) – Bool value that control whether MOSS (True) or MOSNS (False) should be executed.
- **k_neighbors** (*int*) – Amount of nearest neighbors to use in SMOTE if MOSS is used.

Notes

For more details see this paper.

Examples

```
>>> from ITMO_FS.embedded import MOS
>>> from sklearn.linear_model import SGDClassifier
>>> import numpy as np
>>> from sklearn.datasets import make_classification
>>> from sklearn.linear_model import LogisticRegression
>>> dataset = make_classification(n_samples=100, n_features=10,
... n_informative=5, n_redundant=0, weights=[0.85, 0.15], random_state=42,
... shuffle=False)
>>> X, y = np.array(dataset[0]), np.array(dataset[1])
>>> m = MOS(model=SGDClassifier(),
... weight_func=lambda model: np.square(model.coef_).sum(axis=0)).fit(X, y)
>>> m.selected_features_
array([1, 3, 4], dtype=int64)
>>> m = MOS(model=SGDClassifier(), sampling=True,
```

(continues on next page)

(continued from previous page)

```
... weight_func=lambda model: np.square(model.coef_).sum(axis=0)).fit(X, y)
>>> m.selected_features_
array([1, 3, 4, 6], dtype=int64)
```

__init__ (*model*, *weight_func*, *loss='log'*, *seed=42*, *l1_ratio=0.5*, *threshold=0.001*, *epochs=1000*, *alphas=array([0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19])*, *sampling=False*, *k_neighbors=2*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)

Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

3.4 ITMO_FS.hybrid: Hybrid methods

hybrid.FilterWrapperHybrid(filter_, wrapper)	Perform the filter + wrapper hybrid algorithm by first running the filter algorithm on the full dataset, leaving the selected features and running the wrapper algorithm on the cut dataset.
hybrid.Melif(estimator, measure, . . . [, . . .])	MeLiF algorithm.

3.4.1 ITMO_FS.hybrid.FilterWrapperHybrid

class ITMO_FS.hybrid.**FilterWrapperHybrid**(*filter_*, *wrapper*)

Perform the filter + wrapper hybrid algorithm by first running the filter algorithm on the full dataset, leaving the selected features and running the wrapper algorithm on the cut dataset.

Parameters

- **filter** (*object*) – A feature selection model that should have a fit(X, y) method and a **selected_features_** attribute available after fitting.
- **wrapper** (*object*) – A feature selection model that should have a fit(X, y) method, **selected_features_** and **best_score_** attributes available after fitting and a predict(X) method.

Notes

This class doesn't require the first algorithm to be a filter (the only requirements are a fit(X, y) method and a **selected_features_** attribute) but it is recommended to use a fast algorithm first to remove a lot of unnecessary features before processing the resulting dataset with a more time-consuming algorithm (e.g. a wrapper).

Examples

```
>>> import numpy as np
>>> from sklearn.linear_model import LogisticRegression
>>> from ITMO_FS.wrappers.deterministic import BackwardSelection
>>> from ITMO_FS.filters.univariate import UnivariateFilter
>>> from ITMO_FS.hybrid import FilterWrapperHybrid
>>> from sklearn.datasets import make_classification
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=5, n_redundant=0, shuffle=False, random_state=42)
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> filter_ = UnivariateFilter('FRatio', ("K best", 10))
>>> wrapper = BackwardSelection(LogisticRegression(), 5, measure='f1_macro')
```

(continues on next page)

(continued from previous page)

```
>>> model = FilterWrapperHybrid(filter_, wrapper).fit(x, y)
>>> model.selected_features_
array([ 1,  3,  4, 10,  7], dtype=int64)
```

__init__ (*filter_*, *wrapper*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict (*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)

Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

3.4.2 ITMO_FS.hybrid.Melif

class ITMO_FS.hybrid.**Melif**(*estimator*, *measure*, *cutting_rule*, *filter_ensemble*, *delta=0.5*, *points=None*, *seed=42*, *cv=3*)

MeLiF algorithm.

Parameters

- **estimator** (*object*) – A supervised learning estimator that should have a fit(X, y) method and a predict(X) method.
- **measure** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable with signature measure(estimator, X, y) which should return only a single value.
- **cutting_rule** (*string or callable*) – A cutting rule name defined in GLOB_CR or a callable with signature cutting_rule (features), which should return a list features ranked by some rule.
- **filter_ensemble** (*object*) – A filter ensemble (e.g. WeightBased) or a list of filters that will be used to create a WeightBased ensemble.
- **delta** (*float*) – The step in coordinate descent.
- **points** (*array-like*) – An array of starting points in the search.
- **seed** (*int*) – Random seed used to initialize np.random.default_rng().
- **cv** (*int*) – Number of folds in cross-validation.

See also:

For

Examples

```
>>> from ITMO_FS.hybrid import Melif
>>> from ITMO_FS.filters.univariate import UnivariateFilter
>>> from sklearn.datasets import make_classification
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> from sklearn.linear_model import LogisticRegression
```

(continues on next page)

(continued from previous page)

```
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=5, n_redundant=0, shuffle=False, random_state=42)
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> x = KBinsDiscretizer(n_bins=10, encode='ordinal',
... strategy='uniform').fit_transform(x)
>>> filters = [UnivariateFilter('GiniIndex'),
... UnivariateFilter('FechnerCorr'),
... UnivariateFilter('SpearmanCorr'),
... UnivariateFilter('PearsonCorr')]
>>> algo = Melif(LogisticRegression(), 'f1_macro', ("K best", 5),
... filters, delta=0.5).fit(x, y)
>>> algo.selected_features_
array([ 3,  4,  1, 13, 16], dtype=int64)
```

__init__ (*estimator*, *measure*, *cutting_rule*, *filter_ensemble*, *delta=0.5*, *points=None*, *seed=42*, *cv=3*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict (*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

3.5 ITMO_FS.wrappers: Wrapper methods

3.5.1 ITMO_FS.wrappers.deterministic: Deterministic wrapper methods

wrappers.deterministic.AddDelWrapper(...[, ...])	Add-Del feature wrapper.
wrappers.deterministic.BackwardSelection(...)	Backward Selection removes one feature at a time until the number of features to be removed is reached.
wrappers.deterministic.RecursiveElimination(...)	Recursive feature elimination algorithm.
wrappers.deterministic.SequentialForwardSelection(...)	Sequentially add features that maximize the classifying function when combined with the features already used.

ITMO_FS.wrappers.deterministic.AddDelWrapper

class ITMO_FS.wrappers.deterministic.**AddDelWrapper**(*estimator*, *measure*, *cv=3*, *seed=42*, *d=1*)
Add-Del feature wrapper.

Parameters

- **estimator** (*object*) – A supervised learning estimator that should have a fit(X, y) method and a predict(X) method.
- **measure** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable with signature measure(estimator, X, y) which should return only a single value.

- **cv** (*int*) – Number of folds in cross-validation.
- **seed** (*int*) – Seed for python random.
- **d** (*int*) – Amount of consecutive iterations for add ana del procedures that can have decreasing objective function before the algorithm terminates.

See also:

Lecture, p.13

http //www.ccas.ru/voron/download/Modeling.pdf

Examples

```
>>> from sklearn.datasets import make_classification
>>> from sklearn.linear_model import LogisticRegression
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=5, n_redundant=0, shuffle=False, random_state=42)
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> lg = LogisticRegression(solver='lbfgs')
>>> add_del = AddDelWrapper(lg, 'accuracy').fit(x, y)
>>> add_del.selected_features_
array([1, 4, 3], dtype=int64)
```

__init__ (*estimator*, *measure*, *cv=3*, *seed=42*, *d=1*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict(*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.wrappers.deterministic.BackwardSelection

class ITMO_FS.wrappers.deterministic.**BackwardSelection**(*estimator*, *n_features*, *measure*, *cv=3*)
Backward Selection removes one feature at a time until the number of features to be removed is reached. On each step, the best n-1 features out of n are chosen (according to some estimator metric) and the last one is removed.

Parameters

- **estimator** (*object*) – A supervised learning estimator that should have a fit(X, y) method and a predict(X) method.
- **n_features** (*int*) – Number of features to select.
- **measure** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable with signature measure(estimator, X, y) which should return only a single value.
- **cv** (*int*) – Number of folds in cross-validation.

Examples

```
>>> from ITMO_FS.wrappers import BackwardSelection
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import make_classification
>>> import numpy as np
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=5, n_redundant=0, shuffle=False, random_state=42)
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> model = BackwardSelection(LogisticRegression(), 5,
... measure='f1_macro').fit(x, y)
>>> model.selected_features_
array([ 0,  1,  2,  3, 13], dtype=int64)
```

__init__ (*estimator*, *n_features*, *measure*, *cv=3*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict (*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.wrappers.deterministic.RecursiveElimination

class ITMO_FS.wrappers.deterministic.**RecursiveElimination**(*estimator*, *n_features*, *measure*, *weight_func*, *cv=3*)
Recursive feature elimination algorithm.

Parameters

- **estimator** (*object*) – A supervised learning estimator that should have a fit(X, y) method, a predict(X) method and a field corresponding to feature weights.
- **n_features** (*int*) – Number of features to leave.
- **measure** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable with signature measure(estimator, X, y) which should return only a single value.
- **weight_func** (*callable*) – A function to extract weights from the model.
- **cv** (*int*) – Number of folds in cross-validation.

See also:

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V., "Gene selection for cancer classification using support vector machines", Mach. Learn., 46(1-3), 389–422, 2002. https://link.springer.com/article/10.1023/A:1012487302797

Examples

```
>>> from sklearn.datasets import make_classification
>>> from ITMO_FS.wrappers import RecursiveElimination
>>> from sklearn.svm import SVC
>>> import numpy as np
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=4, n_redundant=0, shuffle=False, random_state=42)
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> model = SVC(kernel='linear')
>>> rfe = RecursiveElimination(model, 5, measure='f1_macro',
... weight_func=lambda model: np.square(model.coef_).sum(axis=0)).fit(x, y)
>>> rfe.selected_features_
array([ 0,  1,  2, 11, 19], dtype=int64)
```

__init__ (*estimator*, *n_features*, *measure*, *weight_func*, *cv=3*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict (*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.wrappers.deterministic.SequentialForwardSelection

class ITMO_FS.wrappers.deterministic.**SequentialForwardSelection**(*estimator*, *n_features*, *measure*, *cv=3*)
Sequentially add features that maximize the classifying function when combined with the features already used. #TODO add theory about this method

Parameters

- **estimator** (*object*) – A supervised learning estimator that should have a fit(X, y) method and a predict(X) method.
- **n_features** (*int*) – Number of features to select.
- **measure** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable with signature measure(estimator, X, y) which should return only a single value.
- **cv** (*int*) – Number of folds in cross-validation.

Examples

```
>>> from ITMO_FS.wrappers import SequentialForwardSelection
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import make_classification
>>> import numpy as np
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=5, n_redundant=0, shuffle=False, random_state=42)
```

(continues on next page)

(continued from previous page)

```
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> model = SequentialForwardSelection(LogisticRegression(), 5,
... measure='f1_macro').fit(x, y)
>>> model.selected_features_
array([ 1,  4,  3,  5, 19], dtype=int64)
```

__init__ (*estimator*, *n_features*, *measure*, *cv=3*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict (*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

Deterministic wrapper function

wrappers.deterministic.qpfs_wrapper

ITMO_FS.wrappers.deterministic.qpfs_wrapper

ITMO_FS.wrappers.deterministic.**qpfs_wrapper**()

3.5.2 ITMO_FS.wrappers.randomized: Randomized wrapper methods

wrappers.randomized.HillClimbingWrapper(...)	Hill Climbing algorithm.
wrappers.randomized.SimulatedAnnealing(...)	Simulated Annealing algorithm.
wrappers.randomized.TPhMGWO(estimator, measure)	Grey Wolf optimization with Two-Phase Mutation.

ITMO_FS.wrappers.randomized.HillClimbingWrapper

class ITMO_FS.wrappers.randomized.**HillClimbingWrapper**(*estimator*, *measure*, *seed=42*, *cv=3*)
Hill Climbing algorithm.

Parameters

- **estimator** (*object*) – A supervised learning estimator that should have a fit(X, y) method and a predict(X) method.
- **measure** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable with signature measure(estimator, X, y) which should return only a single value.
- **seed** (*int*) – Random seed used to initialize np.random.default_rng().

- **cv** (*int*) – Number of folds in cross-validation.

Examples

```
>>> from ITMO_FS.wrappers import HillClimbingWrapper
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import make_classification
>>> import numpy as np
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=5, n_redundant=0, shuffle=False, random_state=42)
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> model = HillClimbingWrapper(LogisticRegression(),
... measure='f1_macro').fit(x, y)
>>> model.selected_features_
array([ 0,  1,  2,  3,  4,  6,  7,  9, 11, 13, 14, 15], dtype=int64)
```

__init__ (*estimator*, *measure*, *seed=42*, *cv=3*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict(*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.wrappers.randomized.SimulatedAnnealing

class ITMO_FS.wrappers.randomized.**SimulatedAnnealing**(*estimator*, *measure*, *seed=42*, *iteration_number=100*, *c=1*, *init_number_of_features=None*, *cv=3*)
Simulated Annealing algorithm.

Parameters

- **estimator** (*object*) – A supervised learning estimator that should have a fit(X, y) method and a predict(X) method.
- **measure** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable with signature measure(estimator, X, y) which should return only a single value.
- **seed** (*int*) – Random seed used to initialize np.random.default_rng().
- **iteration_number** (*int*) – Number of iterations of the algorithm.
- **c** (*int*) – A constant that is used to control the rate of feature perturbation.
- **init_number_of_features** (*int*) – The number of features to initialize start features subset with, by default 5-10 percents of features is used.
- **cv** (*int*) – Number of folds in cross-validation.

Notes

For more details see this paper.

Examples

```
>>> from sklearn.datasets import make_classification
>>> from sklearn.linear_model import LogisticRegression
>>> from ITMO_FS.wrappers.randomized import SimulatedAnnealing
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=5, n_redundant=0, shuffle=False, random_state=42)
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> sa = SimulatedAnnealing(LogisticRegression(), measure='f1_macro',
... iteration_number=50).fit(x, y)
>>> sa.selected_features_
array([ 1,  4,  3, 17, 10, 16, 11, 14,  5], dtype=int64)
```

__init__ (*estimator*, *measure*, *seed=42*, *iteration_number=100*, *c=1*, *init_number_of_features=None*, *cv=3*)
Initialize self. See help(type(self)) for accurate signature.

fit (*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict(*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

ITMO_FS.wrappers.randomized.TPhMGWO

class ITMO_FS.wrappers.randomized.**TPhMGWO**(*estimator*, *measure*, *wolf_number=10*, *seed=1*, *alpha=0.5*, *cv=5*, *iteration_number=30*, *mp=0.5*, *binarize='sigmoid'*)
Grey Wolf optimization with Two-Phase Mutation.

Parameters

- **estimator** (*object*) – A supervised learning estimator that should have a fit(X, y) method and a predict(X) method. The original paper suggests to use the k-nearest-neighbors classifier.
- **measure** (*string or callable*) – A standard estimator metric (e.g. 'f1' or 'roc_auc') or a callable with signature measure(estimator, X, y) which should return only a single value.
- **wolf_number** (*int*) – Number of search agents used to find a solution for feature selection problem.
- **seed** (*int*) – Random seed used to initialize np.random.default_rng().
- **alpha** (*float*) – Weight of importance of classification accuracy. Alpha is used in equation that counts fitness as fitness = alpha * score + beta * **|selected_features|** / **|features|** where alpha = 1 - beta.
- **cv** (*int*) – Number of folds in cross-validation.
- **iteration_number** (*int*) – Number of iterations of the algorithm.

- **mp** (*float*) – Mutation probability.
- **binarize** (*str*) – Transformation function to use. Currently only 'tanh' and 'sigmoid' are supported.

Notes

For more details see this paper.

Examples

```
>>> import numpy as np
>>> from sklearn.neighbors import KNeighborsClassifier
>>> from ITMO_FS.wrappers.randomized import TPhMGWO
>>> from sklearn.datasets import make_classification
>>> dataset = make_classification(n_samples=100, n_features=20,
... n_informative=5, n_redundant=0, shuffle=False, random_state=42)
>>> x, y = np.array(dataset[0]), np.array(dataset[1])
>>> tphmgwo = TPhMGWO(KNeighborsClassifier(n_neighbors=7),
... measure='accuracy').fit(x, y)
>>> tphmgwo.selected_features_
array([0, 1, 2, 4], dtype=int64)
```

__init__(*estimator*, *measure*, *wolf_number=10*, *seed=1*, *alpha=0.5*, *cv=5*, *iteration_number=30*, *mp=0.5*, *binarize='sigmoid'*)
Initialize self. See help(type(self)) for accurate signature.

fit(*X*, *y=None*, ***fit_params*)
Fit the algorithm.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.
- **y** (*array-like, shape (n_samples,), optional*) – The class labels.
- **fit_params** (*dict, optional*) – Additional parameters to pass to underlying _fit function.

Returns

Return type Self, i.e. the transformer object.

fit_transform(*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*{array-like, sparse matrix, dataframe} of shape (n_samples, n_features)*) –
- **y** (*ndarray of shape (n_samples,), default=None*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type ndarray array of shape (n_samples, n_features_new)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict(*X*)
Predict class labels for the input data.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The input samples.

Returns **array-like, shape (n_samples,)**

Return type class labels

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)
Transform given data by slicing it with selected features.

Parameters **X** (*array-like, shape (n_samples, n_features)*) – The training input samples.

Returns

Return type Transformed 2D numpy array

CHAPTER 4

Getting started

Information to install, test, and contribute to the package.

CHAPTER 5

User Guide

User guide of ITMO_FS

CHAPTER 6

API

The main documentation. This contains an in-depth description of all algorithms and how to apply them.

CHAPTER 7

API Documentation

The exact API of all functions and classes, as given in the doctring. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

Index

Symbols

A

B

C

D

F

G

T

U

V

W