DiviK

Release 2.5.10

Aug 11, 2020

Topics:

Here you can find a list of documentation topics covered by this page.

CHAPTER 1

Installation

1.1 Docker

The recommended way to use this software is through Docker. This is the most convenient way, if you want to use *divik* application.

To install latest stable version use:

```
docker pull gmrukwa/divik
```

To install specific version, you can specify it in the command, e.g.:

```
docker pull gmrukwa/divik:2.5.10
```

1.2 Python package

Prerequisites for installation of base package:

- Python 3.6 / 3.7 / 3.8
- compiler capable of compiling the native C code

Having prerequisites installed, one can install latest base version of the package:

```
pip install divik
```

or any stable tagged version, e.g.:

```
pip install divik==2.5.10
```

If you want to have compatibility with gin-config, you can install necessary extras with:

```
pip install divik[gin]
```

Note: Remember about *' before '[* and *]* in *zsh* shell.

CHAPTER 2

Running in Docker

2.1 Prerequisites

First of all, you need to have Docker installed. You can proceed with the official instructions:

- Windows
- Ubuntu
- Mac

Under Windows and Mac you need to perform additional configuration steps before running the analysis, since data processing requires additional resources as compared to simple web applications.

1. Right-click the running Docker icon (a whale with squares).
2. Go to *Preferences*
3. Allow Docker to run with all the CPUs and reasonable RAM (at least 16 GB, as much as possible recommended).

Note: Under Ubuntu these steps are not required as Docker runs natively.

2.2 Run the Container

The container is launched with the default Docker syntax, as described here. You can use the following:

- under UNIX:

```
docker run \
    --rm -it \
    --volume $(pwd):/data \
    gmrukwa/divik \
    bash
```

- under Windows:

```
docker run^
    --rm -it^
    --volume %cd%:/data^
    gmrukwa/divik^
    bash
```

In both cases, the directory where the command is ran is mounted to the `\data` directory in the container, so the data and / or configuration is available (see *Data*). `--rm` indicates that the container gets removed after it finishes running. `-it` indicates that the console will get attached to the running container. `gmrukwa/divik` is the image name. Finally, `bash` launches the shell in the container. You can launch any other command there.

2.3 Code

Code of the installed package is available at the */app* directory in the case of need to reinstall.

2.4 Data

Your data should be mounted into the container in the `/data` directory. It is assumed to be the working directory of the Python interpreter. Please remember that all the paths should be relative to this directory or absolute with root at `/data`. This is maintained by the switch `-v $(pwd):/data` under UNIX or `-v %cd%:/data` under Windows.

2.5 I/O Buffering

Python interpreter I/O buffering is turned off by default, so all the outputs appear immediately. Otherwise it would be impossible to track the actual progress of the computations. You can turn this off by setting `PYTHONUNBUFFERED` environment variable to `FALSE`.

CHAPTER 3

Simple Windows Instruction

This is the simplest instruction to run DiviK on Windows.

1. Install Docker (see Install)
2. Create `run_divik.bat` with following content:

```
@echo off
tasklist /FI "IMAGENAME eq Docker Desktop.exe" 2>NUL | find /I /N "Docker Desktop.exe
↪">NUL
if "%ERRORLEVEL%"=="0" (
    echo Docker is running
) ELSE (
    echo Docker is not running, launching - please wait....
    start "" /B "C:\Program Files\Docker\Docker\Docker Desktop.exe"
    timeout 60 /nobreak
)
echo Checking for updates...
docker pull gmrukwa/divik
docker run^
    --rm^
    -it^
    --volume %cd%:/data^
    gmrukwa/divik^
    divik^
    --source /data/data.csv^
    --config /data/divik.json^
    --destination /data/results^
    --verbose
pause
```

1. Put your data into `data.csv`
2. Create `divik.json` starting from such template:

```
{
  "gap_trials": 10,
  "leaf_size": 0.01,
  "max_iter": 100,
  "distance": "correlation",
  "minimal_size": 16,
  "rejection_size": 2,
  "rejection_percentage": null,
  "minimal_features_percentage": 0.01,
  "features_percentage": 0.05,
  "k_max": 50,
  "sample_size": 1000,
  "normalize_rows": true,
  "use_logfilters": true,
  "filter_type": "gmm",
  "n_jobs": -1,
  "random_seed": 0,
  "verbose": true
}
```

1. Adjust the configuration to your needs

Note: Configuration follows the JSON format with fields defined as

here.

1. Double click the `run_divik.bat`

CHAPTER 4

divik package

divik.**plot**(*tree*, *with_size=False*)
Plot visualization of splits.

divik.**reject_split**(*tree: Optional[divik.core._types.DivikResult], rejection_size: int = 0*) → Optional[divik.core._types.DivikResult]
Re-apply rejection condition on known result tree.

CHAPTER 5

core module

divik.core.**Centroids**
alias of numpy.ndarray

divik.core.**Data**
alias of numpy.ndarray

class divik.core.**DivikResult**(*clustering*, *feature_selector*, *merged*, *subregions*)

Attributes

clustering Alias for field number 0

feature_selector Alias for field number 1

merged Alias for field number 2

subregions Alias for field number 3

Methods

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

clustering
Alias for field number 0

count(*value*, */*)
Return number of occurrences of value.

feature_selector
Alias for field number 1

index(*value*, *start=0*, *stop=sys.maxsize*, */*)
Return first index of value.

Raises ValueError if the value is not present.

merged
Alias for field number 2

subregions
Alias for field number 3

divik.core.**IntLabels**
alias of numpy.ndarray

divik.core.**build**(*klass*, ***kwargs*)
Build instance of klass using matching kwargs

divik.core.**context_if**(*condition*, *context*, **args*, ***kwargs*)

divik.core.**normalize_rows**(*data: numpy.ndarray*) → numpy.ndarray

divik.core.**visualize**(*label*, *xy*, *shape=None*)

divik.core.**get_n_jobs**(*n_jobs*)

divik.core.**maybe_pool**(*processes: int = None*, **args*, ***kwargs*)

divik.core.**share**(*array: numpy.ndarray*)

divik.core.**seed**(*seed_: int = 0*)
Crete seeded scope.

divik.core.**seeded**(*wrapped_requires_seed: bool = False*)
Create seeded scope for function call.

Parameters

wrapped_requires_seed: bool, optional, default: False if true, passes seed parameter to the inner function

divik.core.**configurable**(*name_or_fn=None*, *module=None*, *whitelist=None*, *blacklist=None*)
Decorator to make a function or class configurable.

This decorator registers the decorated function/class as configurable, which allows its parameters to be supplied from the global configuration (i.e., set through *bind_parameter* or *parse_config*). The decorated function is associated with a name in the global configuration, which by default is simply the name of the function or class, but can be specified explicitly to avoid naming collisions or improve clarity.

If some parameters should not be configurable, they can be specified in *blacklist*. If only a restricted set of parameters should be configurable, they can be specified in *whitelist*.

The decorator can be used without any parameters as follows:

@config.configurable def some_configurable_function(param1, param2='a default value'):

...

In this case, the function is associated with the name *'some_configurable_function'* in the global configuration, and both *param1* and *param2* are configurable.

The decorator can be supplied with parameters to specify the configurable name or supply a whitelist/blacklist:

@config.configurable('explicit_configurable_name', whitelist='param2') def some_configurable_function(param1, param2='a default value'):

...

In this case, the configurable is associated with the name *'explicit_configurable_name'* in the global configuration, and only *param2* is configurable.

Classes can be decorated as well, in which case parameters of their constructors are made configurable:

@config.configurable class SomeClass(object):

def __init__(self, param1, param2='a default value'): ...

In this case, the name of the configurable is *'SomeClass'*, and both *param1* and *param2* are configurable.

Args:

name_or_fn: A name for this configurable, or a function to decorate (in which case the name will be taken from that function). If not set, defaults to the name of the function/class that is being made configurable. If a name is provided, it may also include module components to be used for disambiguation (these will be appended to any components explicitly specified by *module*).

module: The module to associate with the configurable, to help handle naming collisions. By default, the module of the function or class being made configurable will be used (if no module is specified as part of the name).

whitelist: A whitelisted set of kwargs that should be configurable. All other kwargs will not be configurable. Only one of *whitelist* or *blacklist* should be specified.

blacklist: A blacklisted set of kwargs that should not be configurable. All other kwargs will be configurable. Only one of *whitelist* or *blacklist* should be specified.

Returns: When used with no parameters (or with a function/class supplied as the first parameter), it returns the decorated function or class. When used with parameters, it returns a function that can be applied to decorate the target function or class.

divik.core.**dump_gin_args**(*destination*)

Dump gin-config effective configuration

If you have *gin* extras installed, you can call *dump_gin_args* save effective gin configuration to a file.

divik.core.**parse_gin_args**()

Parse arguments with gin-config

If you have *gin* extras installed, you can call *parse_gin_args* to parse command line arguments or config files to configure your runs.

Command line arguments are used like *–param='DiviK.k_max=50'*. Config files are passed via *–config=path.gin*.

More about format of *.gin* files can be found here: https://github.com/google/gin-config

CHAPTER 6

gin_sklearn_configurables module

Mark scikit-learn classes as configurable

CHAPTER 7

cluster module

Clustering methods

class divik.cluster.**DiviK**(*kmeans*, *fast_kmeans=None*, *distance: str = 'correlation'*, *minimal_size: int = None*, *rejection_size: int = None*, *rejection_percentage: float = None*, *minimal_features_percentage: float = 0.01*, *features_percentage: float = 0.05*, *normalize_rows: bool = None*, *use_logfilters: bool = False*, *filter_type='gmm'*, *n_jobs: int = None*, *verbose: bool = False*)

DiviK clustering

Parameters

kmeans: AutoKMeans A self-tuning KMeans estimator for the purpose of clustering

fast_kmeans: GAPSearch, optional, default: None A self-tuning KMeans estimator for the purpose of stop condition check. If None, the *kmeans* parameter is assumed to be the *GAPSearch* instance.

distance: str, optional, default: 'correlation' The distance metric between points, centroids and for GAP index estimation. One of the distances supported by scipy package.

minimal_size: int, optional, default: None The minimum size of the region (the number of observations) to be considered for any further divisions. When left None, defaults to 0.1% of the training dataset size.

rejection_size: int, optional, default: None Size under which split will be rejected - if a cluster appears in the split that is below rejection_size, the split is considered improper and discarded. This may be useful for some domains (like there is no justification for a 3-cells cluster in biological data). By default, no segmentation is discarded, as careful post-processing provides the same advantage.

rejection_percentage: float, optional, default: None An alternative to `rejection_size`, with the same behavior, but this parameter is related to the training data size percentage. By default, no segmentation is discarded.

minimal_features_percentage: float, optional, default: 0.01 The minimal percentage of features that must be preserved after GMM-based feature selection. By default at least 1% of

features is preserved in the filtration process.

features_percentage: float, optional, default: 0.05 The target percentage of features that are used by fallback percentage filter for 'outlier' filter.

normalize_rows: bool, optional, default: None Whether to normalize each row of the data to the norm of 1. By default, it normalizes rows for correlation metric, does no normalization otherwise.

use_logfilters: bool, optional, default: False Whether to compute logarithm of feature characteristic instead of the characteristic itself. This may improve feature filtering performance, depending on the distribution of features, however all the characteristics (mean, variance) have to be positive for that - filtering will fail otherwise. This is useful for specific cases in biology where the distribution of data may actually require this option for any efficient filtering.

filter_type: {'gmm', 'outlier', 'auto', 'none'}, default: 'gmm'

- 'gmm' - usual Gaussian Mixture Model-based filtering, useful for high

dimensional cases - 'outlier' - robust outlier detection-based filtering, useful for low dimensional cases. In the case of no outliers, percentage-based filtering is applied. - 'auto' - automatically selects between 'gmm' and 'outlier' based on the dimensionality. When more than 250 features are present, 'gmm' is chosen. - 'none' - feature selection is disabled

n_jobs: int, optional, default: None The number of jobs to use for the computation. This works by computing each of the GAP index evaluations in parallel and by making predictions in parallel.

verbose: bool, optional, default: False Whether to report the progress of the computations.

Examples

```
>>> from divik.cluster import DunnDiviK
>>> from sklearn.datasets import make_blobs
>>> X, _ = make_blobs(n_samples=200, n_features=100, centers=20,
...                   random_state=42)
>>> divik = DiviK(distance='euclidean').fit(X)
>>> divik.labels_
array([1, 1, 1, 0, ..., 0, 0], dtype=int32)
>>> divik.predict([[0, ..., 0], [12, ..., 3]])
array([1, 0], dtype=int32)
>>> divik.cluster_centers_
array([[10.,  ...,  2.],
       ...,
       [ 1,  ...,  2.]])
```

Attributes

result_: divik.DivikResult Hierarchical structure describing all the consecutive segmentations.

labels_: Labels of each point

centroids_: array, [n_clusters, n_features] Coordinates of cluster centers. If the algorithm stops before fully converging, these will not be consistent with `labels_`. Also, the distance between points and respective centroids must be captured in appropriate features subspace. This is realized by the `transform` method.

filters_: array, [n_clusters, n_features] Filters that were applied to the feature space on the level that was the final segmentation for a subset.

depth_: int The number of hierarchy levels in the segmentation.

n_clusters_: int The final number of clusters in the segmentation, on the tree leaf level.

paths_: Dict[int, Tuple[int]] Describes how the cluster number corresponds to the path in the tree. Element of the tuple indicates the sub-segment number on each tree level.

reverse_paths_: Dict[Tuple[int], int] Describes how the path in the tree corresponds to the cluster number. For more details see `paths_`.

Methods

fit(X[, y])	Compute DiviK clustering.
fit_predict(X[, y])	Compute cluster centers and predict cluster index for each sample.
fit_transform(X[, y])	Compute clustering and transform X to cluster-distance space.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict the closest cluster each sample in X belongs to.
set_params(**params)	Set the parameters of this estimator.
transform(X)	Transform X to a cluster-distance space.

fit (*X*, *y=None*)
Compute DiviK clustering.

Parameters

X [array-like or sparse matrix, shape=(n_samples, n_features)] Training instances to cluster. It must be noted that the data will be converted to C ordering, which will cause a memory copy if the given data is not C-contiguous.

y [Ignored] not used, present here for API consistency by convention.

fit_predict (*X*, *y=None*)
Compute cluster centers and predict cluster index for each sample.

Convenience method; equivalent to calling fit(X) followed by predict(X).

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to transform.

y [Ignored] not used, present here for API consistency by convention.

Returns

labels [array, shape [n_samples,]] Index of the cluster each sample belongs to.

fit_transform (*X*, *y=None*, ***fit_params*)
Compute clustering and transform X to cluster-distance space.

Equivalent to fit(X).transform(X), but more efficiently implemented.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to transform.

y [Ignored] not used, present here for API consistency by convention.

Returns

X_new [array, shape [n_samples, self.n_clusters_]] X transformed in the new space.

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 [mapping of string to any] Parameter names mapped to their values.

predict(*X*)
Predict the closest cluster each sample in X belongs to.

In the vector quantization literature, *cluster_centers_* is called the code book and each value returned by *predict* is the index of the closest code in the code book.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to predict.

Returns

labels [array, shape [n_samples,]] Index of the cluster each sample belongs to.

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 [object] Estimator instance.

transform(*X*)
Transform X to a cluster-distance space.

In the new space, each dimension is the distance to the cluster centers. Note that even if X is sparse, the array returned by *transform* will typically be dense.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to transform.

Returns

X_new [array, shape [n_samples, self.n_clusters_]] X transformed in the new space.

class divik.cluster.**DunnSearch**(*kmeans: divik.cluster._kmeans._core.KMeans*, *max_clusters: int*, *min_clusters: int = 2*, *method='full'*, *inter='centroid'*, *intra='avg'*, *sample_size=1000*, *n_trials=10*, *seed=42*, *n_jobs: int = 1*, *drop_unfit: bool = False*, *verbose: bool = False*)
Select best number of clusters for k-means

Parameters

kmeans [KMeans] KMeans object to tune

max_clusters: int The maximal number of clusters to form and score.

min_clusters: int, default: 1 The minimal number of clusters to form and score.

method: {'full', 'sampled', 'auto'}, default: 'full' Whether to run full computations or approximate. - full - always computes full Dunn's index, without sampling - sampled - samples the clusters to reduce computational overhead - auto - switches the above methods to provide best performance-quality trade-off.

inter [{'centroid', 'closest'}, default: 'centroid'] How the distance between clusters is computed. For more details see *dunn*.

intra [{'avg', 'furthest'}, default: 'avg'] How the cluster internal distance is computed. For more details see *dunn*.

sample_size [int, default: 1000] Size of the sample used to compute Dunn index in *auto* or *sampled* scenario.

n_trials [int, default: 10] Number of trials to use when computing Dunn index in *auto* or *sampled* scenario.

seed [int, default: 42] Random seed for the reproducibility of subset draws in Dunn *auto* or *sampled* scenario.

n_jobs: int, default: 1 The number of jobs to use for the computation. This works by computing each of the clustering & scoring runs in parallel.

drop_unfit: bool, default: False If True, drops the estimators that did not fit the data.

verbose: bool, default: False If True, shows progress with tqdm.

Attributes

cluster_centers_: array, [n_clusters, n_features] Coordinates of cluster centers.

labels_: Labels of each point.

estimators_: List[KMeans] KMeans instances for n_clusters in range [min_clusters, max_clusters].

scores_: array, [max_clusters - min_clusters + 1,] Array with scores for each estimator.

n_clusters_: int Estimated optimal number of clusters.

best_score_: float Score of the optimal estimator.

best_: KMeans The optimal estimator.

Methods

fit(X[, y])	Compute k-means clustering and estimate optimal number of clusters.
fit_predict(X[, y])	Perform clustering on X and returns cluster labels.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict the closest cluster each sample in X belongs to.
set_params(**params)	Set the parameters of this estimator.
transform(X)	Transform X to a cluster-distance space.

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

Compute k-means clustering and estimate optimal number of clusters.

Parameters

X [array-like or sparse matrix, shape=(n_samples, n_features)] Training instances to cluster. It must be noted that the data will be converted to C ordering, which will cause a memory copy if the given data is not C-contiguous.

y [Ignored] not used, present here for API consistency by convention.

fit_predict(*X*, *y=None*)
Perform clustering on X and returns cluster labels.

Parameters

X [ndarray, shape (n_samples, n_features)] Input data.

y [Ignored] Not used, present for API consistency by convention.

Returns

labels [ndarray, shape (n_samples,)] Cluster labels.

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

predict(*X*)
Predict the closest cluster each sample in X belongs to.

In the vector quantization literature, *cluster_centers_* is called the code book and each value returned by *predict* is the index of the closest code in the code book.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to predict.

Returns

labels [array, shape [n_samples,]] Index of the cluster each sample belongs to.

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 [object] Estimator instance.

transform(*X*)

Transform X to a cluster-distance space.

In the new space, each dimension is the distance to the cluster centers. Note that even if X is sparse, the array returned by *transform* will typically be dense.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to transform.

Returns

X_new [array, shape [n_samples, k]] X transformed in the new space.

class divik.cluster.**GAPSearch**(*kmeans: divik.cluster._kmeans._core.KMeans*, *max_clusters: int*, *min_clusters: int = 1*, *n_jobs: int = 1*, *seed: int = 0*, *n_trials: int = 10*, *sample_size: int = 1000*, *drop_unfit: bool = False*, *verbose: bool = False*)

Select best number of cluters for k-means

Parameters

kmeans [KMeans] KMeans object to tune

max_clusters: int The maximal number of clusters to form and score.

min_clusters: int, default: 1 The minimal number of clusters to form and score.

n_jobs: int, default: 1 The number of jobs to use for the computation. This works by computing each of the clustering & scoring runs in parallel.

seed: int, default: 0 Random seed for generating uniform data sets.

n_trials: int, default: 10 Number of data sets drawn as a reference.

sample_size [int, default: 1000] Size of the sample used for GAP statistic computation. Used only if introduces speedup.

drop_unfit: bool, default: False If True, drops the estimators that did not fit the data.

verbose: bool, default: False If True, shows progress with tqdm.

Attributes

cluster_centers_: array, [n_clusters, n_features] Coordinates of cluster centers.

labels_: Labels of each point.

estimators_: List[KMeans] KMeans instances for n_clusters in range [min_clusters, max_clusters].

scores_: array, [max_clusters - min_clusters + 1, ?] Array with scores for each estimator in each row.

n_clusters_: int Estimated optimal number of clusters.

best_score_: float Score of the optimal estimator.

best_: KMeans The optimal estimator.

Methods

fit(X[, y])	Compute k-means clustering and estimate optimal number of clusters.
fit_predict(X[, y])	Perform clustering on X and returns cluster labels.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict the closest cluster each sample in X belongs to.
set_params(**params)	Set the parameters of this estimator.
transform(X)	Transform X to a cluster-distance space.

fit (*X*, *y=None*)
Compute k-means clustering and estimate optimal number of clusters.

Parameters

X [array-like or sparse matrix, shape=(n_samples, n_features)] Training instances to cluster. It must be noted that the data will be converted to C ordering, which will cause a memory copy if the given data is not C-contiguous.

y [Ignored] not used, present here for API consistency by convention.

fit_predict (*X*, *y=None*)
Perform clustering on X and returns cluster labels.

Parameters

X [ndarray, shape (n_samples, n_features)] Input data.

y [Ignored] Not used, present for API consistency by convention.

Returns

labels [ndarray, shape (n_samples,)] Cluster labels.

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

predict(*X*)

Predict the closest cluster each sample in X belongs to.

In the vector quantization literature, *cluster_centers_* is called the code book and each value returned by *predict* is the index of the closest code in the code book.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to predict.

Returns

labels [array, shape [n_samples,]] Index of the cluster each sample belongs to.

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 [object] Estimator instance.

transform(*X*)

Transform X to a cluster-distance space.

In the new space, each dimension is the distance to the cluster centers. Note that even if X is sparse, the array returned by *transform* will typically be dense.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to transform.

Returns

X_new [array, shape [n_samples, k]] X transformed in the new space.

class divik.cluster.**KMeans**(*n_clusters: int*, *distance: str = 'euclidean'*, *init: str = 'percentile'*, *percentile: float = 95.0*, *leaf_size: Union[int*, *float] = 0.01*, *max_iter: int = 100*, *normalize_rows: bool = False*)

K-Means clustering

Parameters

n_clusters [int] The number of clusters to form as well as the number of centroids to generate.

distance [str, optional, default: 'euclidean'] Distance measure. One of the distances supported by scipy package.

init [{'percentile', 'extreme', 'kdtree', 'kdtree_percentile'}] Method for initialization, defaults to 'percentile':

'percentile' : selects initial cluster centers for k-mean clustering starting from specified percentile of distance to already selected clusters

'extreme': selects initial cluster centers for k-mean clustering starting from the furthest points to already specified clusters

'kdtree': selects initial cluster centers for k-mean clustering starting from centroids of KD-Tree boxes

'kdtree_percentile': selects initial cluster centers for k-means clustering starting from centroids of KD-Tree boxes containing specified percentile. This should be more robust against outliers.

percentile [float, default: 95.0] Specifies the starting percentile for 'percentile' initialization. Must be within range [0.0, 100.0]. At 100.0 it is equivalent to 'extreme' initialization.

leaf_size [int or float, optional (default 0.01)] Desired leaf size in kdtree initialization. When int, the box size will be between *leaf_size* and *2 * leaf_size*. When float, it will be between *leaf_size * n_samples* and *2 * leaf_size * n_samples*

max_iter [int, default: 100] Maximum number of iterations of the k-means algorithm for a single run.

normalize_rows [bool, default: False] If True, rows are translated to mean of 0.0 and scaled to norm of 1.0.

Attributes

cluster_centers_ [array, [n_clusters, n_features]] Coordinates of cluster centers.

labels_ : Labels of each point

Methods

fit(X[, y])	Compute k-means clustering.
fit_predict(X[, y])	Perform clustering on X and returns cluster labels.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict the closest cluster each sample in X belongs to.
set_params(**params)	Set the parameters of this estimator.
transform(X)	Transform X to a cluster-distance space.

fit (*X*, *y=None*)
Compute k-means clustering.

Parameters

X [array-like or sparse matrix, shape=(n_samples, n_features)] Training instances to cluster. It must be noted that the data will be converted to C ordering, which will cause a memory copy if the given data is not C-contiguous.

y [Ignored] not used, present here for API consistency by convention.

fit_predict (*X*, *y=None*)
Perform clustering on X and returns cluster labels.

Parameters

X [ndarray, shape (n_samples, n_features)] Input data.

y [Ignored] Not used, present for API consistency by convention.

Returns

labels [ndarray, shape (n_samples,)] Cluster labels.

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

predict(*X*)
Predict the closest cluster each sample in X belongs to.

In the vector quantization literature, *cluster_centers_* is called the code book and each value returned by *predict* is the index of the closest code in the code book.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to predict.

Returns

labels [array, shape [n_samples,]] Index of the cluster each sample belongs to.

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 [object] Estimator instance.

transform(*X*)
Transform X to a cluster-distance space.

In the new space, each dimension is the distance to the cluster centers. Note that even if X is sparse, the array returned by *transform* will typically be dense.

Parameters

X [{array-like, sparse matrix}, shape = [n_samples, n_features]] New data to transform.

Returns

X_new [array, shape [n_samples, k]] X transformed in the new space.

CHAPTER 8

feature_selection module

Unsupervised feature selection methods

class divik.feature_selection.**SelectorMixin**

Transformer mixin that performs feature selection given a support mask

This mixin provides a feature selector implementation with *transform* and *inverse_transform* functionality given an implementation of *_get_support_mask*.

Methods

fit_transform(X[, y])	Fit to data, then transform it.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
transform(X)	Reduce X to the selected features.

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

get_support(*indices=False*)

Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)

Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

transform(*X*)

Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

class divik.feature_selection.**StatSelectorMixin**

Transformer mixin that performs feature selection given a support mask

This mixin provides a feature selector implementation with *transform* and *inverse_transform* functionality given that *selected_* is specified during *fit*.

Additionally, provides a *_to_characteristics* and *_to_raw* implementations given *stat*, optionally *use_log* and *preserve_high*.

Methods

fit_transform(X[, y])	Fit to data, then transform it.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
transform(X)	Reduce X to the selected features.

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

get_support(*indices=False*)

Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)

Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

transform(*X*)

Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

class divik.feature_selection.**NoSelector**

Dummy selector to use when no selection is supposed to be made.

Methods

fit(X[, y])	Pass data forward
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
set_params(**params)	Set the parameters of this estimator.
transform(X)	Reduce X to the selected features.

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

Pass data forward

Parameters

X [{array-like, sparse matrix}, shape (n_samples, n_features)] Sample vectors to pass.

y [any] Ignored. This parameter exists only for compatibility with sklearn.pipeline.Pipeline.

Returns

self

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

get_support(*indices=False*)
Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)
Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

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 [object] Estimator instance.

transform(*X*)

Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

class divik.feature_selection.**EximsSelector**

Select features based on their spatial distribution

Preserves features that yield biologically plausible structures.

References

Wijetunge, Chalini D., et al. "EXIMS: an improved data analysis pipeline based on a new peak picking method for EXploring Imaging Mass Spectrometry data." Bioinformatics 31.19 (2015): 3198-3206. https://academic.oup.com/bioinformatics/article/31/19/3198/212150

Methods

fit(X[, y, xy])	Learn data-driven feature thresholds from X.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
set_params(**params)	Set the parameters of this estimator.
transform(X)	Reduce X to the selected features.

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

Learn data-driven feature thresholds from X.

Parameters

X [{array-like, sparse matrix}, shape (n_samples, n_features)] Sample vectors from which to compute feature characteristic.

y [any] Ignored. This parameter exists only for compatibility with sklearn.pipeline.Pipeline.

xy [array-like, shape (n_samples, 2)] Spatial coordinates of the samples. Expects integers, indices over am image.

Returns

self

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

get_support(*indices=False*)
Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)
Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

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 [object] Estimator instance.

transform(*X*)

Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

class divik.feature_selection.**GMMSelector**(*stat: str*, *use_log: bool = False*, *n_candidates: int = None*, *min_features: int = 1*, *min_features_rate: float = 0.0*, *preserve_high: bool = True*, *max_components: int = 10*)

Feature selector that removes low- or high- mean or variance features

Gaussian Mixture Modeling is applied to the features' characteristics and components are obtained. Crossing points of the components are considered candidate thresholds. Out of these up to `n_candidates` components are removed in such a way that at least `min_features` or `min_features_rate` features are retained.

This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

Parameters

stat: {'mean', 'var'}

Kind of statistic to be computed out of the feature.

use_log: bool, optional, default: False Whether to use the logarithm of feature characteristic instead of the characteristic itself. This may improve feature filtering performance, depending on the distribution of features, however all the characteristics (mean, variance) have to be positive for that - filtering will fail otherwise. This is useful for specific cases in biology where the distribution of data may actually require this option for any efficient filtering.

n_candidates: int, optional, default: None How many candidate thresholds to use at most. `0` preserves all the features (all candidate thresholds are discarded), `None` allows to remove all but one component (all candidate thresholds are retained). Negative value means to discard up to all but `-n_candidates` candidates, e.g. `-1` will retain at least two components (one candidate threshold is removed).

min_features: int, optional, default: 1 How many features must be preserved. Candidate thresholds are tested against this value, and if they retain less features, less conservative thresholds is selected.

min_features_rate: float, optional, default: 0.0 Similar to `min_features` but relative to the input data features number.

preserve_high: bool, optional, default: True Whether to preserve the high-characteristic features or low-characteristic ones.

max_components: int, optional, default: 10 The maximum number of components used in the GMM decomposition.

Examples

import divik._utils >>> import numpy as np

```
>>> import divik.feature_selection as fs
>>> divik._utils.seed(42)
>>> labels = np.concatenate([30 * [0] + 20 * [1] + 30 * [2] + 40 * [3]])
>>> data = labels * 5 + np.random.randn(*labels.shape)
>>> fs.GMMSelector('mean').fit_transform(data)
array([[14.78032811 15.35711257 ... 15.75193303]])
>>> fs.GMMSelector('mean', preserve_high=False).fit_transform(data)
array([[ 0.49671415 -0.1382643  ... -0.29169375]])
>>> fs.GMMSelector('mean', n_discard=-1).fit_transform(data)
array([[10.32408397  9.61491772 ... 15.75193303]])
```

Attributes

vals_: array, shape (n_features,)

Computed characteristic of each feature.

threshold_: float Threshold value to filter the features by the characteristic.

raw_threshold_: float Threshold value mapped back to characteristic space (no logarithm, etc.)

selected_: array, shape (n_features,) Vector of binary selections of the informative features.

Methods

fit(X[, y])	Learn data-driven feature thresholds from X.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
set_params(**params)	Set the parameters of this estimator.
transform(X)	Reduce X to the selected features.

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

Learn data-driven feature thresholds from X.

Parameters

X [{array-like, sparse matrix}, shape (n_samples, n_features)] Sample vectors from which to compute feature characteristic.

y [any] Ignored. This parameter exists only for compatibility with sklearn.pipeline.Pipeline.

Returns

self

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

get_support(*indices=False*)
Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)
Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

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 [object] Estimator instance.

transform(*X*)
Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

divik.feature_selection.**huberta_outliers**(*v*)

M. Huberta, E.Vandervierenb (2008) An adjusted boxplot for skewed distributions, Computational Statistics and Data Analysis 52 (2008) 5186–5201

Parameters

v: array-like An array to filter outlier from.

Returns

Binary vector indicating all the outliers.

class divik.feature_selection.**OutlierSelector**(*stat: str*, *use_log: bool = False*, *keep_outliers: bool = False*)

Feature selector that removes outlier features w.r.t. mean or variance

Huberta's outlier detection is applied to the features' characteristics and the outlying features are removed.

This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

Parameters

stat: {'mean', 'var'} Kind of statistic to be computed out of the feature.

use_log: bool, optional, default: False Whether to use the logarithm of feature characteristic instead of the characteristic itself. This may improve feature filtering performance, depending on the distribution of features, however all the characteristics (mean, variance) have to be positive for that - filtering will fail otherwise. This is useful for specific cases in biology where the distribution of data may actually require this option for any efficient filtering.

keep_outliers: bool, optional, default: False When True, keeps outliers instead of inlier features.

Attributes

vals_: array, shape (n_features,) Computed characteristic of each feature.

selected_: array, shape (n_features,) Vector of binary selections of the informative features.

Methods

fit(X[, y])	Learn data-driven feature thresholds from X.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
set_params(**params)	Set the parameters of this estimator.
transform(X)	Reduce X to the selected features.

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

Learn data-driven feature thresholds from X.

Parameters

X [{array-like, sparse matrix}, shape (n_samples, n_features)] Sample vectors from which to compute feature characteristic.

y [any] Ignored. This parameter exists only for compatibility with sklearn.pipeline.Pipeline.

Returns

self

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

get_support(*indices=False*)
Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)
Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

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 [object] Estimator instance.

transform(*X*)
Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

class divik.feature_selection.**PercentageSelector**(*stat: str*, *use_log: bool = False*, *keep_top: bool = True*, *p: float = 0.2*)
Feature selector that removes / preserves top some percent of features

This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

Parameters

stat: {'mean', 'var'} Kind of statistic to be computed out of the feature.

use_log: bool, optional, default: False Whether to use the logarithm of feature characteristic instead of the characteristic itself. This may improve feature filtering performance, depending on the distribution of features, however all the characteristics (mean, variance) have to be positive for that - filtering will fail otherwise. This is useful for specific cases in biology where the distribution of data may actually require this option for any efficient filtering.

keep_top: bool, optional, default: True When True, keeps features with highest value of the characteristic.

p: float, optional, default: 0.2 Rate of features to keep.

Attributes

vals_: array, shape (n_features,) Computed characteristic of each feature.

threshold_: float Value of the threshold used for filtering

selected_: array, shape (n_features,) Vector of binary selections of the informative features.

Methods

fit(X[, y])	Learn data-driven feature thresholds from X.
fit_transform(X[, y])	Fit to data, then transform it.

Continued on next page

Table 7 – continued from previous page

get_params([deep])	Get parameters for this estimator.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
set_params(**params)	Set the parameters of this estimator.
transform(X)	Reduce X to the selected features.

fit(*X*, *y=None*)
Learn data-driven feature thresholds from X.

Parameters

X [{array-like, sparse matrix}, shape (n_samples, n_features)] Sample vectors from which to compute feature characteristic.

y [any] Ignored. This parameter exists only for compatibility with sklearn.pipeline.Pipeline.

Returns

self

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

get_support(*indices=False*)
Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)
Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

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 [object] Estimator instance.

transform(*X*)
Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

class divik.feature_selection.**HighAbundanceAndVarianceSelector**(*use_log: bool = False, min_features: int = 1, min_features_rate: float = 0.0, max_components: int = 10*)
Feature selector that removes low-mean and low-variance features

Exercises GMMSelector to filter out the low-abundance noise features and select high-variance informative features.

This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

Parameters

use_log: bool, optional, default: False

Whether to use the logarithm of feature characteristic instead of the characteristic itself. This may improve feature filtering performance, depending on the distribution of features, however all the characteristics (mean, variance) have to be positive for that - filtering will fail otherwise. This is useful for specific cases in biology where the distribution of data may actually require this option for any efficient filtering.

min_features: int, optional, default: 1 How many features must be preserved.

min_features_rate: float, optional, default: 0.0 Similar to `min_features` but relative to the input data features number.

max_components: int, optional, default: 10 The maximum number of components used in the GMM decomposition.

Examples

import divik._utils >>> import numpy as np

```
>>> import divik.feature_selection as fs
>>> divik._utils.seed(42)
>>> # Data in this case must be carefully crafted
>>> labels = np.concatenate([30 * [0] + 20 * [1] + 30 * [2] + 40 * [3]])
>>> data = np.vstack(100 * [labels * 10.])
>>> data += np.random.randn(*data.shape)
>>> sub = data[:, :-40]
>>> sub += 5 * np.random.randn(*sub.shape)
>>> # Label 0 has low abundance but high variance
>>> # Label 3 has low variance but high abundance
>>> # Label 1 and 2 has not-lowest abundance and high variance
>>> selector = fs.HighAbundanceAndVarianceSelector().fit(data)
>>> selector.transform(labels.reshape(1,-1))
array([[1 1 1 1 1 ...2 2 2]])
```

Attributes

abundance_selector_: GMMSelector

Selector used to filter out the noise component.

variance_selector_: GMMSelector Selector used to filter out the non-informative features.

selected_: array, shape (n_features,) Vector of binary selections of the informative features.

Methods

fit(X[, y])	Learn data-driven feature thresholds from X.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
set_params(**params)	Set the parameters of this estimator.
transform(X)	Reduce X to the selected features.

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

Learn data-driven feature thresholds from X.

Parameters

X [{array-like, sparse matrix}, shape (n_samples, n_features)] Sample vectors from which to compute feature characteristic.

y [any] Ignored. This parameter exists only for compatibility with sklearn.pipeline.Pipeline.

Returns

self

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

get_support(*indices=False*)
Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)
Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

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 [object] Estimator instance.

transform(*X*)

Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

class divik.feature_selection.**OutlierAbundanceAndVarianceSelector**(*use_log: bool = False, min_features_rate: float = 0.01, p: float = 0.2*)

Methods

fit(X[, y])	Learn data-driven feature thresholds from X.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
get_support([indices])	Get a mask, or integer index, of the features selected
inverse_transform(X)	Reverse the transformation operation
set_params(**params)	Set the parameters of this estimator.
transform(X)	Reduce X to the selected features.

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

Learn data-driven feature thresholds from X.

Parameters

X [{array-like, sparse matrix}, shape (n_samples, n_features)] Sample vectors from which to compute feature characteristic.

y [any] Ignored. This parameter exists only for compatibility with sklearn.pipeline.Pipeline.

Returns

self

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

get_support(*indices=False*)
Get a mask, or integer index, of the features selected

Parameters

indices [boolean (default False)] If True, the return value will be an array of integers, rather than a boolean mask.

Returns

support [array] An index that selects the retained features from a feature vector. If *indices* is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for retention. If *indices* is True, this is an integer array of shape [# output features] whose values are indices into the input feature vector.

inverse_transform(*X*)
Reverse the transformation operation

Parameters

X [array of shape [n_samples, n_selected_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_original_features]] *X* with columns of zeros inserted where features would have been removed by *transform()*.

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 [object] Estimator instance.

transform(*X*)
Reduce X to the selected features.

Parameters

X [array of shape [n_samples, n_features]] The input samples.

Returns

X_r [array of shape [n_samples, n_selected_features]] The input samples with only the selected features.

divik.feature_selection.**make_specialized_selector**(*name*, *n_features*, ***kwargs*)
Create a selector by name (gmm, outlier or auto)

CHAPTER 9

feature_extraction module

class divik.feature_extraction.**KneePCA**(*whiten: bool = False*, *refit: bool = False*)

Principal component analysis (PCA) with knee method

PCA with automated components selection based on knee method over cumulative explained variance. Remaining components are discarded.

Parameters

whiten [bool, optional (default False)] When True (False by default) the *pca_.components_* vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.

refit [bool, optional (default False)] When True (False by default) the *pca_* is re-fit with the smaller number of components. This could reduce memory footprint, but requires training fitting PCA.

Attributes

pca_ [PCA] Fit PCA estimator.

n_components_ [int] The number of selected components.

Methods

fit(X[, y])	Fit the model from data in X.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
inverse_transform(X)	Transform data back to its original space.
set_params(**params)	Set the parameters of this estimator.
transform(X[, y])	Apply dimensionality reduction to X.

fit(*X*, *y=None*)
Fit the model from data in X.

Parameters

X [array-like, shape (n_samples, n_features)] Training vector, where n_samples is the number of samples and n_features is the number of features.

Y: Ignored.

Returns

self [object] Returns the instance itself.

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 [numpy array of shape [n_samples, n_features]] Training set.

y [numpy array of shape [n_samples]] Target values.

****fit_params** [dict] Additional fit parameters.

Returns

X_new [numpy array of shape [n_samples, n_features_new]] Transformed array.

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 [mapping of string to any] Parameter names mapped to their values.

inverse_transform(*X*)
Transform data back to its original space.

In other words, return an input X_original whose transform would be X.

Parameters

X [array-like, shape (n_samples, n_components)] New data, where n_samples is the number of samples and n_components is the number of components.

Returns

X_original array-like, shape (n_samples, n_features)

Notes

If whitening is enabled, inverse_transform will compute the exact inverse operation, which includes reversing whitening.

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 [object] Estimator instance.

transform(*X*, *y=None*)

Apply dimensionality reduction to X.

X is projected on the first principal components previously extracted from a training set.

Parameters

X [array-like, shape (n_samples, n_features)] New data, where n_samples is the number of samples and n_features is the number of features.

Returns

X_new [array-like, shape (n_samples, n_components)]

Examples

```
>>> import numpy as np
>>> from divik.feature_extraction import KneePCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = KneePCA(refit=True)
>>> pca.fit(X)
KneePCA(refit=True)
>>> pca.transform(X) # doctest: +SKIP
```

class divik.feature_extraction.**LocallyAdjustedRbfSpectralEmbedding**(*distance: str = 'euclidean', n_components=2, random_state=None, eigen_solver: str = None, n_neighbors: int = None, n_jobs: int = 1*)

Spectral embedding for non-linear dimensionality reduction.

Forms an affinity matrix given by the specified function and applies spectral decomposition to the corresponding graph laplacian. The resulting transformation is given by the value of the eigenvectors for each data point.

Note : Laplacian Eigenmaps is the actual algorithm implemented here.

Parameters

distance [{'braycurtis', 'canberra', 'chebyshev', 'cityblock',]

'correlation', 'cosine', 'dice', 'euclidean', 'hamming', 'jaccard',

'kulsinski', 'mahalanobis', 'atching', 'minkowski', 'rogerstanimoto',

'russellrao', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule'} Distance measure, defaults to 'euclidean'. These are the distances supported by scipy package.

n_components [integer, default: 2] The dimension of the projected subspace.

random_state [int, RandomState instance or None, optional, default: None] A pseudo random number generator used for the initialization of the lobpcg eigenvectors. If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*. Used when `solver` == 'amg'.

eigen_solver [{None, 'arpack', 'lobpcg', or 'amg'}] The eigenvalue decomposition strategy to use. AMG requires pyamg to be installed. It can be faster on very large, sparse problems, but may also lead to instabilities.

n_neighbors [int, default] Number of nearest neighbors for nearest_neighbors graph building.

n_jobs [int, optional (default = 1)] The number of parallel jobs to run. If `-1`, then the number of jobs is set to the number of CPU cores.

References

- A Tutorial on Spectral Clustering, 2007 Ulrike von Luxburg http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.9323
- On Spectral Clustering: Analysis and an algorithm, 2001 Andrew Y. Ng, Michael I. Jordan, Yair Weiss http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.8100
- Normalized cuts and image segmentation, 2000 Jianbo Shi, Jitendra Malik http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.160.2324

Attributes

embedding_ [array, shape = (n_samples, n_components)] Spectral embedding of the training matrix.

Methods

fit(X[, y])	Fit the model from data in X.
fit_transform(X[, y])	Fit the model from data in X and transform X.
get_params([deep])	Get parameters for this estimator.
save(destination)	Save embedding to a directory
set_params(**params)	Set the parameters of this estimator.

transform	

fit (*X*, *y=None*)
Fit the model from data in X.

Parameters

X [array-like, shape (n_samples, n_features)] Training vector, where n_samples is the num-

ber of samples and n_features is the number of features.

Y: Ignored.

Returns

self [object] Returns the instance itself.

fit_transform(*X*, *y=None*)
Fit the model from data in X and transform X.

Parameters

X [array-like, shape (n_samples, n_features)] Training vector, where n_samples is the number of samples and n_features is the number of features.

Y: Ignored.

Returns

X_new [array-like, shape (n_samples, n_components)]

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 [mapping of string to any] Parameter names mapped to their values.

save(*destination: str*)
Save embedding to a directory

Parameters

destination [str] Directory to save the embedding.

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 [object] Estimator instance.

transform(*X*, *y=None*)

CHAPTER 10

sampler module

class divik.sampler.**BaseSampler**

Base class for all the samplers

Sampler is Pool-safe, i.e. can simply store a dataset. It will not be serialized by pickle when going to another process, if handled properly.

Before you spawn a pool, a data must be moved to a module-level variable. To simplify that process a contract has been prepared. You open a context and operate within a context:

with sampler.parallel() as sampler_

with Pool(initializer=sampler_.initializer,

initargs=sampler_.initargs) as pool:

pool.map(**sampler_**.get_sample, range(10))

Keep in mind, that __iter__ and fit are not accessible in parallel context. __iter__ would yield the same values independently in all the workers. Now it needs to be done consciously and in well-though manner. fit could lead to a non-predictable behaviour. If you need the original sampler, you can get a clone (not fit to the data).

Methods

fit(X[, y])	Fit sampler to data
get_params([deep])	Get parameters for this estimator.
get_sample(seed)	Return specific sample
parallel()	Create parallel context for the sampler to operate
set_params(**params)	Set the parameters of this estimator.

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

Fit sampler to data

It's a base for both supervised and unsupervised samplers.

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 [mapping of string to any] Parameter names mapped to their values.

get_sample(*seed*)
Return specific sample

Following assumptions should be met: a) sampler.get_sample(x) == sampler.get_sample(x) b) x != y should yield sampler.get_sample(x) != sampler.get_sample(y)

Parameters

seed [int] The seed to use to draw the sample

Returns

sample [array_like, (*self.shape_)] Returns the drawn sample

parallel()
Create parallel context for the sampler to operate

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 [object] Estimator instance.

class divik.sampler.**ParallelSampler**(*sampler: divik.sampler._core.BaseSampler*)
Helper class for sharing the sampler functionality

Attributes

initargs

Methods

clone()	Clones the original sampler
get_sample(seed)	Return specific sample

initializer	

clone()
Clones the original sampler

get_sample(*seed*)
Return specific sample

initargs

initializer(**args*)

class divik.sampler.**UniformSampler**(*n_rows: int = None*, *n_samples: int = None*)

Samples uniformly from the boundaries of the data

Parameters

n_rows [int, optional (default None)] Allows to limit the number of rows in the drawn samples

n_samples [int, optional (default None)] Allows to limit the number of samples when iterating

Attributes

shape_ [(n_rows, n_cols)] Shape of the drawn samples

scaler_ [MinMaxScaler] Scaler ensuring the proper ranges

Methods

fit(X[, y])	Fit the model from data in X.
get_params([deep])	Get parameters for this estimator.
get_sample(seed)	Return specific sample
parallel()	Create parallel context for the sampler to operate
set_params(**params)	Set the parameters of this estimator.

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

Fit the model from data in X.

Parameters

X [array-like, shape (n_samples, n_features)] Training vector, where n_samples is the number of samples and n_features is the number of features.

Y: Ignored.

Returns

self [UniformSampler] Returns the instance itself.

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 [mapping of string to any] Parameter names mapped to their values.

get_sample(*seed*)

Return specific sample

Parameters

seed [int] The seed to use to draw the sample

Returns

sample [array_like, (***self.shape_**)] Returns the drawn sample

parallel()
Create parallel context for the sampler to operate

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 [object] Estimator instance.

class divik.sampler.**UniformPCASampler**(*n_rows: int = None*, *n_samples: int = None*, *whiten: bool = False*, *refit: bool = False*, *pca: str = 'knee'*)
Rotation-invariant uniform sampling

Parameters

n_rows [int, optional (default None)] Allows to limit the number of rows in the drawn samples

n_samples [int, optional (default None)] Allows to limit the number of samples when iterating

whiten [bool, optional (default False)] When True (False by default) the *pca_.components_* vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.

refit [bool, optional (default False)] When True (False by default) the *pca_* is re-fit with the smaller number of components. This could reduce memory footprint, but requires training fitting PCA.

pca: {'knee', 'full'}, default 'knee' Specifies whether to train full or knee PCA.

Attributes

pca_ [KneePCA or PCA] PCA transform which provided rotation-invariance

sampler_ [UniformSampler] Sampler from the transformed distribution

Methods

fit(X[, y])	Fit the model from data in X.
get_params([deep])	Get parameters for this estimator.
get_sample(seed)	Return specific sample
parallel()	Create parallel context for the sampler to operate
set_params(**params)	Set the parameters of this estimator.

fit(*X*, *y=None*)
Fit the model from data in X.

PCA is fit to estimate the rotation and UniformSampler is fit to transformed data.

Parameters

X [array-like, shape (n_samples, n_features)] Training vector, where n_samples is the number of samples and n_features is the number of features.

Y: Ignored.

Returns

self [UniformPCASampler] Returns the instance itself.

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 [mapping of string to any] Parameter names mapped to their values.

get_sample(*seed*)
Return specific sample

Sample is generated from transformed distribution and transformed back to the original space.

Parameters

seed [int] The seed to use to draw the sample

Returns

sample [array_like, (**self.shape_*)] Returns the drawn sample

parallel()
Create parallel context for the sampler to operate

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 [object] Estimator instance.

class divik.sampler.**StratifiedSampler**(*n_rows: Union[int, float] = 100*, *n_samples: int = None*)
Sample the original data preserving proportions of groups

Parameters

n_rows [int or float, optional (default 10000)] Allows to limit the number of rows in the drawn samples. If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the sample. If int, represents the absolute number of rows.

n_samples [int, optional (default None)] Allows to limit the number of samples when iterating

Attributes

X_ [array_like, shape (n_rows, n_features)] Data to sample from

y_ [array_like, shape (n_rows,)] Group labels

Methods

fit(X, y)	Fit the model from data in X.
get_params([deep])	Get parameters for this estimator.
get_sample(seed)	Return specific sample
parallel()	Create parallel context for the sampler to operate
set_params(**params)	Set the parameters of this estimator.

fit (*X*, *y*)
Fit the model from data in X.

Both inputs are preserved inside to sample from the data.

Parameters

X [array-like, shape (n_rows, n_features)] Training vector, where n_rows is the number of rows and n_features is the number of features.

y: array-like, shape (n_rows,)

Returns

self [StratifiedSampler] Returns the instance itself.

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 [mapping of string to any] Parameter names mapped to their values.

get_sample (*seed*)
Return specific sample

Sample is drawn from the set of existing rows. A proportion of gorups should be more-or-less the same, depending on the size of the sample.

Parameters

seed [int] The seed to use to draw the sample

Returns

sample [array_like, (***self.shape_**)] Returns the drawn sample

parallel ()
Create parallel context for the sampler to operate

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 [object] Estimator instance.

CHAPTER 11

Indices and tables

- genindex
- modindex

Python Module Index

d

Index

G

H

I

K

L

M

N

O

P

R

S

T

U

V