DiviK

Release 2.3.5

Dec 23, 2019

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, since it requires *MATLAB Compiler Runtime* and more dependencies.

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:2.3.5
```

1.2 Python package

Prerequisites for installation of base package:

- Python 3.5

These are required for using *divik* application and GMM-based filtering:

- MATLAB Compiler Runtime, version 2016b or newer, installed to default path
- compiled package with legacy code

Installation process may be clearer with insight into Docker images used for application deployment:

- python_mcr image - installs MCR r2016b onto Python 3.5 image
- python_msi image - installs compiled legacy code onto MCR image
- divik image - installs DiviK software onto legacy code image

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==2.3.5
```

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,
  "distance_percentile": 99.0,
  "max_iter": 100,
  "distance": "correlation",
  "minimal_size": 16,
  "rejection_size": 2,
  "minimal_features_percentage": 0.01,
  "fast_kmeans_iter": 10,
  "k_max": 10,
  "normalize_rows": true,
  "use_logfilters": true,
  "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 <https://github.com/gmrukwa/divik/blob/master/divik/_cli/divik.md>‘_.

1. Double click the `run_divik.bat`

CHAPTER 4

divik package

divik.**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

class divik.**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()	
index()	Raises ValueError if the value is not present.

clustering

Alias for field number 0

count()

feature_selector

Alias for field number 1

index()

Raises ValueError if the value is not present.

merged
Alias for field number 2

subregions
Alias for field number 3

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

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

CHAPTER 5

cluster module

Clustering methods

class divik.cluster.**AutoKMeans**(*max_clusters: int*, *min_clusters: int = 1*, *n_jobs: int = 1*, *method: str = 'dunn'*, *distance: str = 'euclidean'*, *init: str = 'percentile'*, *percentile: float = 95.0*, *max_iter: int = 100*, *normalize_rows: bool = False*, *gap=None*, *verbose: bool = False*)

K-Means clustering with automated selection of number of clusters

Parameters

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.

method: {'dunn', 'gap'} The method to select the best number of clusters.

'dunn' : computes score that relates dispersion inside a cluster to distances between clusters. Never selects 1 cluster.

'gap' : compares dispersion of a clustering to a dispersion in grouping of a reference uniformly distributed dataset

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

init: {'percentile' or 'extreme'} 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

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.

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.

gap: dict Configuration of GAP statistic in a form of dict.

max_iter: int, default: 10 Maximal number of iterations KMeans will do for computing statistic.

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

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

correction: bool, default: True If True, the correction is applied and the first feasible solution is selected. Otherwise the globally maximal GAP is used.

Default: {'max_iter': 10, 'seed': 0, 'trials': 10, 'correction': True}

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

fit(*self*, *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(*self*, *X*, *y=None*)
Performs clustering on X and returns cluster labels.

Parameters

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

Returns

y [ndarray, shape (n_samples,)] cluster labels

fit_transform(*self*, *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.

Returns

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

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

Parameters

deep [boolean, optional] 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(*self*, *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(*self*, ***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.

Returns

self

transform(*self*, *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.**DiviK**(*gap_trials: int = 10*, *distance_percentile: float = 99.0*, *max_iter: int = 100*, *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*, *fast_kmeans_iter: int = 10*, *k_max: int = 10*, *normalize_rows: bool = None*, *use_logfilters: bool = False*, *filter_type='gmm'*, *keep_outliers=False*, *n_jobs: int = None*, *random_seed: int = 0*, *verbose: bool = False*)

DiviK clustering

Parameters

gap_trials: int, optional, default: 10 The number of random dataset draws to estimate the GAP index for the clustering quality assessment.

distance_percentile: float, optional, default: 99.0 The percentile of the distance between points and their closest centroid. 100.0 would simply select the furthest point from all the centroids found already. Lower value provides better robustness against outliers. Too low value reduces the capability to detect centroid candidates during initialization.

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

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.

fast_kmeans_iter: int, optional, default: 10 Maximum number of iterations of the k-means algorithm for a single run during computation of the GAP index. Decreased with respect to the max_iter, as GAP index requires multiple segmentations to be evaluated.

k_max: int, optional, default: 10 Maximum number of clusters evaluated during the auto-tuning process. From 1 up to k_max clusters are tested per evaluation.

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

keep_outliers: bool, optional, default: False When *filter_type* is *'outlier'*, this will switch feature selection to outliers-preserving mode (inlier features are removed).

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.

random_seed: int, optional, default: 0 Seed to initialize the random number generator.

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

Examples

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

fit (*self*, *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 (*self*, *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 (*self*, *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(*self*, *deep=True*)
Get parameters for this estimator.

Parameters

deep [boolean, optional] 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(*self*, *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(*self*, ***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.

Returns

self

transform(*self*, *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.**KMeans**(*n_clusters: int*, *distance: str = 'euclidean'*, *init: str = 'percentile'*, *percentile: float = 95.0*, *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' or 'extreme'}] 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

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.

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

fit (*self*, *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 (*self*, *X*, *y=None*)
Performs clustering on X and returns cluster labels.

Parameters

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

Returns

y [ndarray, shape (n_samples,)] cluster labels

fit_transform (*self*, *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.

Returns

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

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

Parameters

deep [boolean, optional] 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(*self*, *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(*self*, ***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.

Returns

self

transform(*self*, *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 6

feature_selection module

Unsupervised feature selection methods

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

fit_transform(*self*, *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.

Returns

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

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

fit(*self*, *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(*self*, *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.

Returns

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

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

Parameters

deep [boolean, optional] 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(*self*, *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(*self*, *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(*self*, ***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.

Returns

self

transform(*self*, *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 numpy as np
>>> import divik.feature_selection as fs
>>> np.random.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]])
```

(continues on next page)

(continued from previous page)

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

fit (*self*, *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 (*self*, *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.

Returns

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

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

Parameters

deep [boolean, optional] 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(*self*, *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(*self*, *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(*self*, ***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.

Returns

self

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

fit(*self*, *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(*self*, *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.

Returns

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

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

Parameters

deep [boolean, optional] 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(*self*, *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(*self*, *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(*self*, ***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.

Returns

self

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

fit(*self*, *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(*self*, *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.

Returns

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

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

Parameters

deep [boolean, optional] 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(*self*, *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(*self*, *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(*self*, ***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.

Returns

self

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

fit (*self*, *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 (*self*, *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.

Returns

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

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

Parameters

deep [boolean, optional] 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 (*self*, *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(*self*, *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(*self*, ***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.

Returns

self

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

fit(*self*, *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(*self*, *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.

Returns

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

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

Parameters

deep [boolean, optional] 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(*self*, *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(*self*, *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(*self*, ***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.

Returns

self

transform(*self*, *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.

CHAPTER 7

Indices and tables

- genindex
- modindex

Python Module Index

d

Index

H

I

K

M

N

O

P

R

S

T