sklearn.grid_search.GridSearchCV

class sklearn.grid_search.**GridSearchCV**(*estimator*, *param_grid*, *scoring=None*, *fit_params=None*, *n_jobs=1*, *iid=True*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score='raise'*) [source]

Exhaustive search over specified parameter values for an estimator.

Important members are fit, predict.

GridSearchCV implements a “fit” and a “score” method. It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Read more in the User Guide.

Parameters: **estimator** : estimator object.

A object of that type is instantiated for each grid point. This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_grid : dict or list of dictionaries

Dictionary with parameters names (string) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

scoring : string, callable or None, default=None

A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`. If `None`, the `score` method of the estimator is used.

fit_params : dict, optional

Parameters to pass to the fit method.

n_jobs : int, default=1

Number of jobs to run in parallel.

Changed in version 0.17: Upgraded to joblib 0.9.3.

pre_dispatch : int, or string, optional

Controls the number of jobs that get dispatched during parallel execution. Reducing this number can be useful to avoid an explosion of memory consumption when more jobs get dispatched than CPUs can process. This parameter can be:

- None, in which case all the jobs are immediately created and spawned. Use this for lightweight and fast-running jobs, to avoid delays due to on-demand spawning of the jobs
- An int, giving the exact number of total jobs that are spawned
- A string, giving an expression as a function of n_jobs, as in '2*n_jobs'

iid : boolean, default=True

If True, the data is assumed to be identically distributed across the folds, and the loss minimized is the total loss per sample, and not the mean loss across the folds.

cv : int, cross-validation generator or an iterable, optional

Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 3-fold cross-validation,

- integer, to specify the number of folds.
- An object to be used as a cross-validation generator.
- An iterable yielding train/test splits.

For integer/None inputs, if `y` is binary or multiclass, **`StratifiedKFold`** used. If the estimator is a classifier or if `y` is neither binary nor multiclass, **`KFold`** is used.

Refer User Guide for the various cross-validation strategies that can be used here.

refit : boolean, default=True

Refit the best estimator with the entire dataset. If “False”, it is impossible to make predictions using this GridSearchCV instance after fitting.

verbose : integer

Controls the verbosity: the higher, the more messages.

error_score : ‘raise’ (default) or numeric

«

Value to assign to the score if an error occurs in estimator fitting. If set to ‘raise’, the error is raised. If a numeric value is given, FitFailedWarning is raised. This parameter does not affect the refit step, which will always raise the error.

Attributes: **grid_scores_** : list of named tuples

Contains scores for all parameter combinations in param_grid. Each entry corresponds to one parameter setting. Each named tuple has the attributes:

- `parameters`, a dict of parameter settings
- `mean_validation_score`, the mean score over the cross-validation folds
- `cv_validation_scores`, the list of scores for each fold

best_estimator_ : estimator

Estimator that was chosen by the search, i.e. estimator which gave highest score (or

smallest loss if specified) on the left out data. Not available if refit=False.

best_score_ : float

Score of best_estimator on the left out data.

best_params_ : dict

Parameter setting that gave the best results on the hold out data.

scorer_ : function

Scorer function used on the held out data to choose the best parameters for the model.

Previous

«

See also:

`ParameterGrid`
generates all the combinations of a an hyperparameter grid.

`sklearn.cross_validation.train_test_split`
utility function to split the data into a development set usable for fitting a GridSearchCV instance and an evaluation set for its final evaluation.

`sklearn.metrics.make_scorer`
Make a scorer from a performance metric or loss function.

Notes

The parameters selected are those that maximize the score of the left out data, unless an explicit score is passed in which case it is used instead.

If *n_jobs* was set to a value higher than one, the data is copied for each point in the grid (and not *n_jobs* times). This is done for efficiency reasons if individual jobs take very little time, but may raise errors if the dataset is large and not enough memory is available. A workaround in this case is to set *pre_dispatch*. Then, the memory is copied only *pre_dispatch* many times. A reasonable value for *pre_dispatch* is *2 * n_jobs*.

Previous

Examples

```
>>> from sklearn import svm, grid_search, datasets
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svr = svm.SVC()
>>> clf = grid_search.GridSearchCV(svr, parameters)
>>> clf.fit(iris.data, iris.target)
...
GridSearchCV(cv=None, error_score=...,
       estimator=SVC(C=1.0, cache_size=..., class_weight=..., coef0=...,
                     decision_function_shape=None, degree=..., gamma=...,
                     kernel='rbf', max_iter=-1, probability=False,
                     random_state=None, shrinking=True, tol=...,
                     verbose=False),
       fit_params={}, iid=..., n_jobs=1,
       param_grid=..., pre_dispatch=..., refit=...,
       scoring=..., verbose=...)
```

Methods

`decision_function`(X)	Call decision_function on the estimator with the best found parameters.
`fit`(X[, y])	Run fit with all sets of parameters.
`get_params`([deep])	Get parameters for this estimator.
`inverse_transform`(Xt)	Call inverse_transform on the estimator with the best found parameters.
`predict`(X)	Call predict on the estimator with the best found parameters.
`predict_log_proba`(X)	Call predict_log_proba on the estimator with the best found parameters.
`predict_proba`(X)	Call predict_proba on the estimator with the best found parameters.
`score`(X[, y])	Returns the score on the given data, if the estimator has been refit.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X)	Call transform on the estimator with the best found parameters.

__init__(*estimator*, *param_grid*, *scoring=None*, *fit_params=None*, *n_jobs=1*, *iid=True*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score='raise'*) [source]

decision_function(*X*) [source]

Call decision_function on the estimator with the best found parameters.

Only available if `refit=True` and the underlying estimator supports `decision_function`.

Parameters: **X** : indexable, length n_samples

Must fulfill the input assumptions of the underlying estimator.

Previous

`fit`(*X*, *y=None*) [source]

Run fit with all sets of parameters.

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 : array-like, shape = [n_samples] or [n_samples, n_output], optional

Target relative to X for classification or regression; None for unsupervised learning.

«

`get_params`(*deep=True*) [source]

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.

inverse_transform(*Xt*) [source]

Call inverse_transform on the estimator with the best found parameters.

Only available if the underlying estimator implements `inverse_transform` and `refit=True`.

Parameters: **Xt** : indexable, length n_samples

Must fulfill the input assumptions of the underlying estimator.

Previous

predict(*X*) [source]

Call predict on the estimator with the best found parameters.

Only available if `refit=True` and the underlying estimator supports `predict`.

Parameters: **X** : indexable, length n_samples

«

Must fulfill the input assumptions of the underlying estimator.

predict_log_proba(*X*) [source]

Call predict_log_proba on the estimator with the best found parameters.

Only available if `refit=True` and the underlying estimator supports `predict_log_proba`.

Parameters: **X** : indexable, length n_samples

Must fulfill the input assumptions of the underlying estimator.

predict_proba(*X*) [source]

Call predict_proba on the estimator with the best found parameters.

Only available if `refit=True` and the underlying estimator supports `predict_proba`.

Parameters: **X** : indexable, length n_samples

Must fulfill the input assumptions of the underlying estimator.

Previous

score(*X*, *y=None*) [source]

Returns the score on the given data, if the estimator has been refit.

This uses the score defined by `scoring` where provided, and the `best_estimator_.score` method otherwise.

Parameters: **X** : array-like, shape = [n_samples, n_features]

Input data, where n_samples is the number of samples and n_features is the number of features.

«

y : array-like, shape = [n_samples] or [n_samples, n_output], optional

Target relative to X for classification or regression; None for unsupervised learning.

Returns: **score** : float

Notes

- The long-standing behavior of this method changed in version 0.16.
- It no longer uses the metric provided by `estimator.score` if the `scoring` parameter was set when fitting.

set_params(***params*) [source]

Set the parameters of this estimator.

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

Returns: **self** :

`transform`(X) [source]

Call transform on the estimator with the best found parameters.

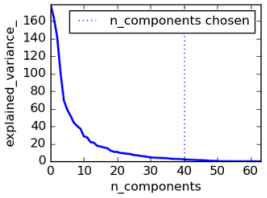
Only available if the underlying estimator supports `transform` and `refit=True`.

Parameters: **X** : indexable, length n_samples

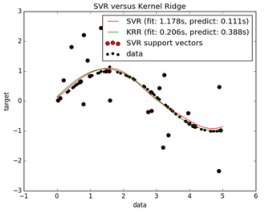
Must fulfill the input assumptions of the underlying estimator.

Examples using `sklearn.grid_search.GridSearchCV`

Concatenating multiple feature extraction methods



Pipelining: chaining a PCA and a logistic regression



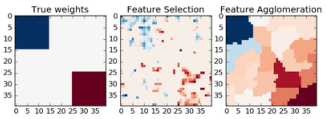
Comparison of kernel ridge regression and SVR

Faces recognition example using eigenfaces and SVMs

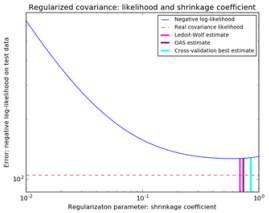
Previous

«

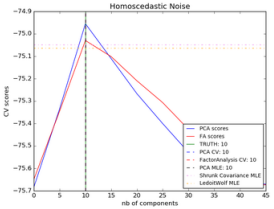
Previous



Feature agglomeration vs. univariate selection



Shrinkage covariance estimation: LedoitWolf vs OAS and max-likelihood

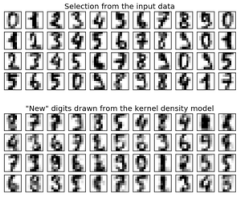


Model selection with Probabilistic PCA and Factor Analysis (FA)

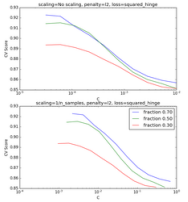
Parameter estimation using grid search with cross-validation

« Comparing randomized search and grid search for hyperparameter estimation

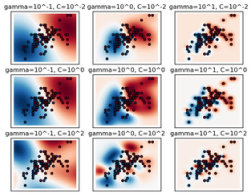
Sample pipeline for text feature extraction and evaluation



Kernel Density Estimation



Scaling the regularization parameter for SVCs



RBF SVM parameters