sklearn.feature_selection.RFECV

class `sklearn.feature_selection.RFECV`(*estimator*, *step=1*, *min_features_to_select=1*, *cv=None*, *scoring=None*, *verbose=0*, *n_jobs=None*)
[source]

Feature ranking with recursive feature elimination and cross-validated selection of the best number of features.

See glossary entry for cross-validation estimator.

Read more in the User Guide.

Parameters:

estimator : *object*
A supervised learning estimator with a `fit` method that provides information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

step : *int or float, optional (default=1)*
If greater than or equal to 1, then `step` corresponds to the (integer) number of features to remove at each iteration. If within (0.0, 1.0), then `step` corresponds to the percentage (rounded down) of features to remove at each iteration. Note that the last iteration may remove fewer than `step` features in order to reach `min_features_to_select`.

min_features_to_select : *int, (default=1)*
The minimum number of features to be selected. This number of features will always be scored, even if the difference between the original feature count and `min_features_to_select` isn't divisible by `step`.

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 5-fold cross-validation,
- integer, to specify the number of folds.
- CV splitter,
- An iterable yielding (train, test) splits as arrays of indices.

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

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

> *Changed in version 0.22:* `cv` default value of None changed from 3-fold to 5-fold.

scoring : *string, callable or None, optional, (default=None)*
A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`.

verbose : *int, (default=0)*
Controls verbosity of output.

n_jobs : *int or None, optional (default=None)*
Number of cores to run in parallel while fitting across folds. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

Attributes:

n_features_ : *int*
The number of selected features with cross-validation.

support_ : *array of shape [n_features]*
The mask of selected features.

ranking_ : *array of shape [n_features]*
The feature ranking, such that `ranking_[i]` corresponds to the ranking position of the i-th feature. Selected (i.e., estimated best) features are assigned rank 1.

grid_scores_ : *array of shape [n_subsets_of_features]*
The cross-validation scores such that `grid_scores_[i]` corresponds to the CV score of the i-th subset of features.

Toggle Menu _ : ***object***

The external estimator fit on the reduced dataset.

See also:

RFE

Recursive feature elimination

Notes

The size of `grid_scores_` is equal to `ceil((n_features - min_features_to_select) / step) + 1`, where step is the number of features removed at each iteration.

Allows NaN/Inf in the input if the underlying estimator does as well.

References

R6f4d61ceb411-1 Guyon, I., Weston, J., Barnhill, S., & Vapnik, V., "Gene selection for cancer classification using support vector machines", Mach. Learn., 46(1-3), 389–422, 2002.

Examples

The following example shows how to retrieve the a-priori not known 5 informative features in the Friedman #1 dataset.

```
>>> from sklearn.datasets import make_friedman1
>>> from sklearn.feature_selection import RFECV
>>> from sklearn.svm import SVR
>>> X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)
>>> estimator = SVR(kernel="linear")
>>> selector = RFECV(estimator, step=1, cv=5)
>>> selector = selector.fit(X, y)
>>> selector.support_
array([ True,  True,  True,  True,  True, False, False, False, False,
       False])
>>> selector.ranking_
array([1, 1, 1, 1, 1, 6, 4, 3, 2, 5])
```

Methods

`decision_function`(self, X)	Compute the decision function of X.
`fit`(self, X, y[, groups])	Fit the RFE model and automatically tune the number of selected
`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
`predict`(self, X)	Reduce X to the selected features and then predict using the
`predict_log_proba`(self, X)	Predict class log-probabilities for X.
`predict_proba`(self, X)	Predict class probabilities for X.
`score`(self, X, y)	Reduce X to the selected features and then return the score of the
`set_params`(self, **params)	Set the parameters of this estimator.
`transform`(self, X)	Reduce X to the selected features.

`__init__`(*self, estimator, step=1, min_features_to_select=1, cv=None, scoring=None, verbose=0, n_jobs=None*) [source]

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

`decision_function`(*self, X*) [source]

Compute the decision function of X.

Parameters:

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

The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

Returns:

score : *array, shape = [n_samples, n_classes] or [n_samples]*

The decision function of the input samples. The order of the classes corresponds to that in the attribute classes_. Regression and binary classification produce an array of shape [n_samples].

Toggle Menu *y, groups=None*) [source]

Fit the RFE model and automatically tune the number of selected
features.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_features)*
Training vector, where `n_samples` is the number of samples and `n_features` is the total number of features.

y : *array-like of shape (n_samples,)*
Target values (integers for classification, real numbers for regression).

groups : *array-like of shape (n_samples,) or None*
Group labels for the samples used while splitting the dataset into train/test set. Only used in conjunction with a "Group" cv instance (e.g., `GroupKFold`).

`fit_transform`(*self*, *X*, *y=None*, ***fit_params*) [source]

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`(*self*, *deep=True*) [source]

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`(*self*, *indices=False*) [source]

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.

Toggle Menu `ansform`(*self*, *X*) [source]

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`](#).

`predict`(*self*, *X*) [source]

Reduce X to the selected features and then predict using the
underlying estimator.

Parameters:

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

Returns:

y : *array of shape [n_samples]*
The predicted target values.

`predict_log_proba`(*self*, *X*) [source]

Predict class log-probabilities for X.

Parameters:

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

Returns:

p : *array of shape (n_samples, n_classes)*
The class log-probabilities of the input samples. The order of the classes corresponds to that in the attribute classes_.

`predict_proba`(*self*, *X*) [source]

Predict class probabilities for X.

Parameters:

X : *{array-like or sparse matrix} of shape (n_samples, n_features)*
The input samples. Internally, it will be converted to `dtype=np.float32` and if a sparse matrix is provided to a sparse `csr_matrix`.

Returns:

p : *array of shape (n_samples, n_classes)*
The class probabilities of the input samples. The order of the classes corresponds to that in the attribute classes_.

`score`(*self*, *X*, *y*) [source]

Reduce X to the selected features and then return the score of the
underlying estimator.

Parameters:

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

y : *array of shape [n_samples]*
The target values.

`set_params`(*self*, ***params*) [source]

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`(*self*, *X*) [source]

Reduce X to the selected features.

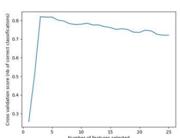
Parameters:

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.

Examples using `sklearn.feature_selection.RFECV`



[Recursive feature elimination with cross-validation]

Toggle Menu