

# sklearn.linear\_model.LinearRegression

class sklearn.linear\_model.LinearRegression(\*, fit\_intercept=True, normalize='deprecated', copy\_X=True, n\_jobs=None, positive=False) [source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients  $w = (w_1, \dots, w_p)$  to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

## Parameters:

**fit\_intercept : bool, default=True**

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

**normalize : bool, default=False**

This parameter is ignored when `fit_intercept` is set to False. If True, the regressors  $X$  will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use [StandardScaler](#) before calling `fit` on an estimator with `normalize=False`.

Deprecated since version 1.0: normalize was deprecated in version 1.0 and will be removed in 1.2.

**copy\_X : bool, default=True**

If True,  $X$  will be copied; else, it may be overwritten.

**n\_jobs : int, default=None**

The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly `n_targets > 1` and secondly  $X$  is sparse or if `positive` is set to `True`. `None` means 1 unless in a [joblib.parallel\\_backend](#) context. `-1` means using all processors. See [Glossary](#) for more details.

**positive : bool, default=False**

When set to `True`, forces the coefficients to be positive. This option is only supported for dense arrays.

New in version 0.24.

## Attributes:

**coef\_ : array of shape (n\_features, ) or (n\_targets, n\_features)**

Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit ( $y$  2D), this is a 2D array of shape  $(n\_targets, n\_features)$ , while if only one target is passed, this is a 1D array of length `n_features`.

**rank\_ : int**

Rank of matrix  $x$ . Only available when  $x$  is dense.

**singular\_ : array of shape (min(X, y),)**

Singular values of  $x$ . Only available when  $x$  is dense.

**intercept\_ : float or array of shape (n\_targets,)**

Independent term in the linear model. Set to 0.0 if `fit_intercept = False`.

**n\_features\_in\_ : int**

Number of features seen during [fit](#).

New in version 0.24.

**feature\_names\_in\_ : ndarray of shape (n\_features\_in\_,)**

Names of features seen during [fit](#). Defined only when  $x$  has feature names that are all strings.

New in version 1.0.

## See also:

[Ridge](#)

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients with l2 on.

Toggle Menu

Lasso

The Lasso is a linear model that estimates sparse coefficients with l1 regularization.

ElasticNet

Elastic-Net is a linear regression model trained with both l1 and l2 -norm regularization of the coefficients.

Notes

From the implementation point of view, this is just plain Ordinary Least Squares (scipy.linalg.lstsq) or Non Negative Least Squares (scipy.optimize.nnls) wrapped as a predictor object.

Examples

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Methods

<a href="#">fit</a> (X, y[, sample_weight])	Fit linear model.
<a href="#">get_params</a> ([deep])	Get parameters for this estimator.
<a href="#">predict</a> (X)	Predict using the linear model.
<a href="#">score</a> (X, y[, sample_weight])	Return the coefficient of determination of the prediction.
<a href="#">set_params</a> (**params)	Set the parameters of this estimator.

fit(X, y, sample\_weight=None)

[source]

Fit linear model.

Parameters:

- X : {array-like, sparse matrix} of shape (n\_samples, n\_features)**  
Training data.
- y : array-like of shape (n\_samples,) or (n\_samples, n\_targets)**  
Target values. Will be cast to X's dtype if necessary.
- sample\_weight : array-like of shape (n\_samples,), default=None**  
Individual weights for each sample.

*New in version 0.17:* parameter *sample\_weight* support to LinearRegression.

Returns:

**self : object**  
Fitted Estimator.

get\_params(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:

Toggle Menu

**dict**  
Parameter names mapped to their values.

predict(X)

[\[source\]](#)

Predict using the linear model.

Parameters:

**X : array-like or sparse matrix, shape (n\_samples, n\_features)**

Samples.

Returns:

**C : array, shape (n\_samples,)**

Returns predicted values.

score(X, y, sample\_weight=None)

[\[source\]](#)

Return the coefficient of determination of the prediction.

The coefficient of determination  $R^2$  is defined as  $(1 - \frac{u}{v})$ , where  $u$  is the residual sum of squares  $((y_{\text{true}} - y_{\text{pred}})** 2).sum()$  and  $v$  is the total sum of squares  $((y_{\text{true}} - y_{\text{true}.mean()}) ** 2).sum()$ . The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of  $y$ , disregarding the input features, would get a  $R^2$  score of 0.0.

Parameters:

**X : array-like of shape (n\_samples, n\_features)**

Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape  $(n_{\text{samples}}, n_{\text{samples\_fitted}})$ , where  $n_{\text{samples\_fitted}}$  is the number of samples used in the fitting for the estimator.

**y : array-like of shape (n\_samples,) or (n\_samples, n\_outputs)**

True values for  $x$ .

**sample\_weight : array-like of shape (n\_samples,), default=None**

Sample weights.

Returns:

**score : float**

$R^2$  of `self.predict(X)` wrt.  $y$ .

Notes

The  $R^2$  score used when calling `score` on a regressor uses `multioutput='uniform_average'` from version 0.23 to keep consistent with default value of [r2\\_score](#). This influences the `score` method of all the multioutput regressors (except for [MultiOutputRegressor](#)).

set\_params(\*\*params)

[\[source\]](#)

Set the parameters of this estimator.

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

Parameters:

**\*\*params : dict**

Estimator parameters.

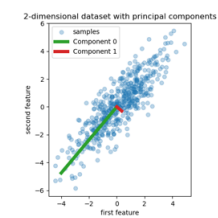
Returns:

**self : estimator instance**

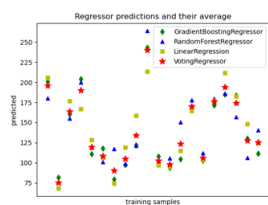
Estimator instance.

Examples using `sklearn.linear_model.LinearRegression`

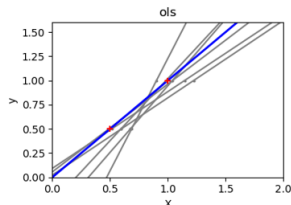
Toggle Menu



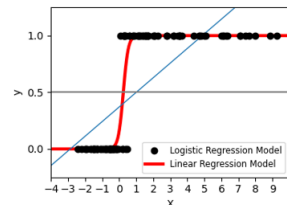
[Principal Component Regression vs Partial Least Squares Regression](#)



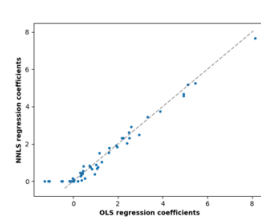
[Plot individual and voting regression predictions](#)



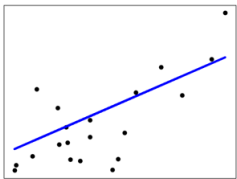
[Ordinary Least Squares and Ridge Regression Variance](#)



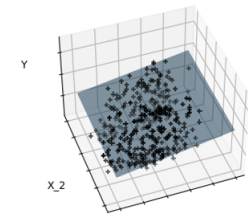
[Logistic function](#)



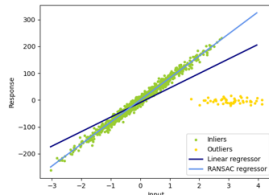
[Non-negative least squares](#)



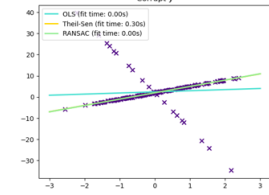
[Linear Regression Example](#)



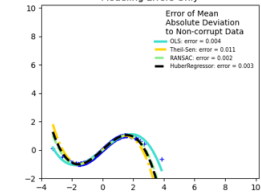
[Sparsity Example: Fitting only features 1 and 2](#)



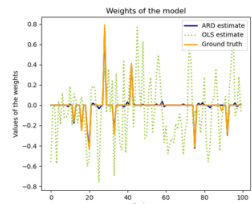
[Robust linear model estimation using RANSAC](#)



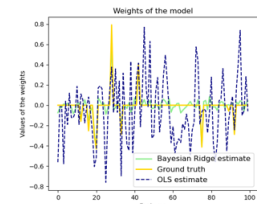
[Theil-Sen Regression](#)



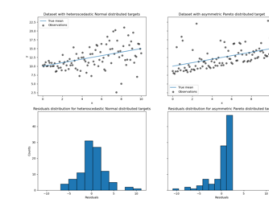
[Robust linear estimator fitting](#)



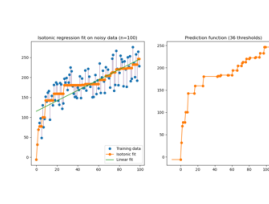
[Automatic Relevance Determination Regression \(ARD\)](#)



[Bayesian Ridge Regression](#)



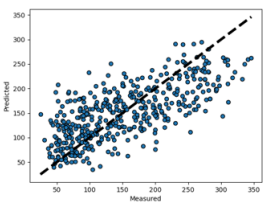
[Quantile regression](#)



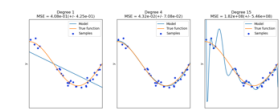
[Isotonic Regression](#)



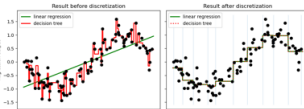
[Face completion with a multi-output estimators](#)



[Plotting Cross-Validated Predictions](#)



[Underfitting vs. Overfitting](#)



[Using KBinsDiscretizer to discretize continuous features](#)