LinearRegression

class sklearn.linear_model.**LinearRegression**(**, fit_intercept=True, copy_X=True, n_jobs=None, positive=False*) [source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) 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).

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.

> *Added 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.

> *Added 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.

> *Added 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 regularization.

`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_
np.float64(3.0...)
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

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.

> *Added in version 0.17:* parameter *sample_weight* support to LinearRegression.

Returns:

self : *object*

Fitted Estimator.

get_metadata_routing() [source]

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns:

routing : *MetadataRequest*

A `MetadataRequest` encapsulating routing information.

`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:

params : *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_true - y_pred)** 2).sum()` and v is the total sum of squares `((y_true - y_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_samples, n_samples_fitted)`, where `n_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)` w.r.t. `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_fit_request(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → LinearRegression [source]

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note

This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters:

sample_weight : *str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*

Metadata routing for `sample_weight` parameter in `fit`.

Returns:

self : *object*

The updated object.

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.

set_score_request(*, sample_weight: bool | None | str = '$UNCHANGED$') → LinearRegression [source]

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note

This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters:

sample_weight : *str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*

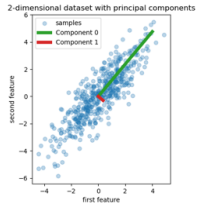
Metadata routing for `sample_weight` parameter in `score`.

Returns:

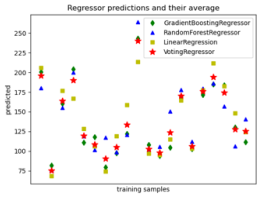
self : *object*

The updated object.

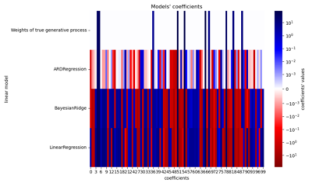
Gallery examples



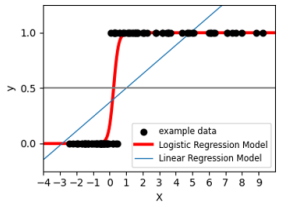
Principal Component Regression vs Partial Least Squares Regression



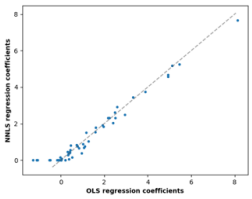
Plot individual and voting regression predictions



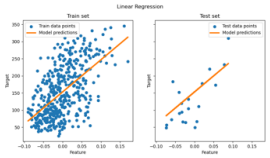
Comparing Linear Bayesian Regressors



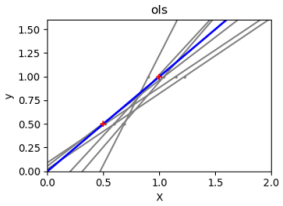
Logistic function



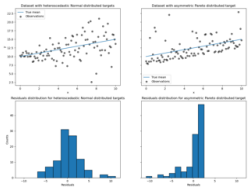
Non-negative least squares



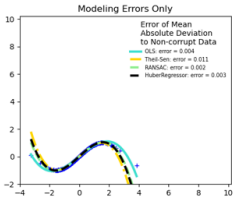
Ordinary Least Squares Example



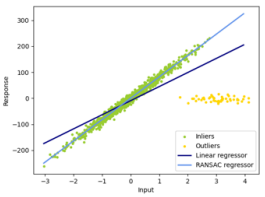
Ordinary Least Squares and Ridge Regression Variance



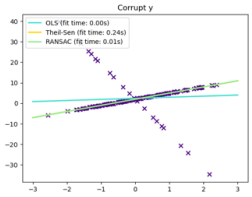
Quantile regression



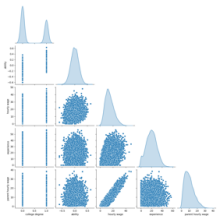
Robust linear estimator fitting



Robust linear model estimation using RANSAC

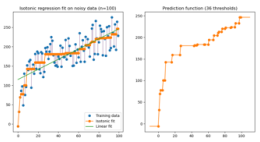


Theil-Sen Regression



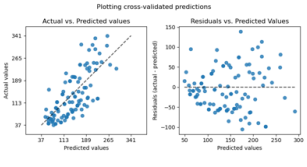
Failure of Machine Learning to infer causal effects

Face completion with a multi-output estimators

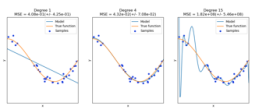


Isotonic Regression

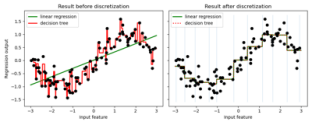
Metadata Routing



Plotting Cross-Validated Predictions



Underfitting vs. Overfitting



Using KBinsDiscretizer to discretize