sklearn.linear_model.Ridge

class sklearn.linear_model.Ridge(*alpha=1.0*, ***, *fit_intercept=True*, *normalize='deprecated'*, *copy_X=True*, *max_iter=None*, *tol=0.001*, *solver='auto'*, *positive=False*, *random_state=None*) [source]

Linear least squares with l2 regularization.

Minimizes the objective function:

```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape (n_samples, n_targets)).

Read more in the User Guide.

Parameters:

alpha : *{float, ndarray of shape (n_targets,)}, default=1.0*
Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. Alpha corresponds to `1 / (2C)` in other linear models such as `LogisticRegression` or `LinearSVC`. If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number.

fit_intercept : *bool, default=True*
Whether to fit the intercept for this model. If set to false, no intercept will be used in calculations (i.e. `X` and `y` are 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.

max_iter : *int, default=None*
Maximum number of iterations for conjugate gradient solver. For 'sparse_cg' and 'lsqr' solvers, the default value is determined by scipy.sparse.linalg. For 'sag' solver, the default value is 1000. For 'lbfgs' solver, the default value is 15000.

tol : *float, default=1e-3*
Precision of the solution.

solver : *{'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga', 'lbfgs'}, default='auto'*
Solver to use in the computational routines:

- 'auto' chooses the solver automatically based on the type of data.
- 'svd' uses a Singular Value Decomposition of X to compute the Ridge coefficients. More stable for singular matrices than 'cholesky'.
- 'cholesky' uses the standard scipy.linalg.solve function to obtain a closed-form solution.

- 'sparse_cg' uses the conjugate gradient solver as found in scipy.sparse.linalg.cg. As an iterative algorithm, this solver is more appropriate than 'cholesky' for large-scale data (possibility to set `tol` and `max_iter`).
- 'lsqr' uses the dedicated regularized least-squares routine scipy.sparse.linalg.lsqr. It is the fastest and uses an iterative procedure.
- 'sag' uses a Stochastic Average Gradient descent, and 'saga' uses its improved, unbiased version named SAGA. Both methods also use an iterative procedure, and are often faster than other solvers when both n_samples and n_features are large. Note that 'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.
- 'lbfgs' uses L-BFGS-B algorithm implemented in `scipy.optimize.minimize`. It can be used only when `positive` is True.

All last six solvers support both dense and sparse data. However, only 'sag', 'sparse_cg', and 'lbfgs' support sparse input when `fit_intercept` is True.

New in version 0.17: Stochastic Average Gradient descent solver.

New in version 0.19: SAGA solver.

positive : *bool, default=False*
When set to `True`, forces the coefficients to be positive. Only 'lbfgs' solver is supported in this case.

random_state : *int, RandomState instance, default=None*
Used when `solver` == 'sag' or 'saga' to shuffle the data. See Glossary for details.

New in version 0.17: `random_state` to support Stochastic Average Gradient.

Attributes:

coef_ : *ndarray of shape (n_features,) or (n_targets, n_features)*
Weight vector(s).

intercept_ : *float or ndarray of shape (n_targets,)*
Independent term in decision function. Set to 0.0 if `fit_intercept = False`.

n_iter_ : *None or ndarray of shape (n_targets,)*
Actual number of iterations for each target. Available only for sag and lsqr solvers. Other solvers will return None.

New in version 0.17.

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:

`RidgeClassifier`
Ridge classifier.

`RidgeCV`
Ridge regression with built-in cross validation.

`KernelRidge`
Kernel ridge regression combines ridge regression with the kernel trick.

Examples

```
>>> from sklearn.linear_model import Ridge
>>> import numpy as np
>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.randn(n_samples)
>>> X = rng.randn(n_samples, n_features)
>>> clf = Ridge(alpha=1.0)
>>> clf.fit(X, y)
Ridge()
```

Methods

fit(X, y[, sample_weight])	Fit Ridge regression model.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict using the linear model.
score(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
set_params(**params)	Set the parameters of this estimator.

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

Fit Ridge regression model.

Parameters:

X : *{ndarray, sparse matrix} of shape (n_samples, n_features)*
Training data.

y : *ndarray of shape (n_samples,) or (n_samples, n_targets)*
Target values.

sample_weight : *float or ndarray of shape (n_samples,), default=None*
Individual weights for each sample. If given a float, every sample will have the same weight.

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:

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

predict(*X*) [source]

Predict using the linear model.

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)` 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.

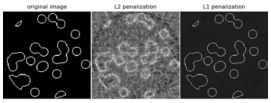
Parameters:

****params : *dict***
Estimator parameters.

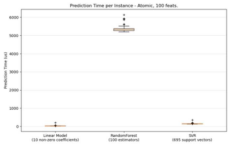
Returns:

self : *estimator instance*
or instance.

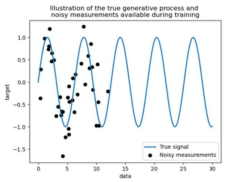
Examples using `sklearn.linear_model.Ridge`



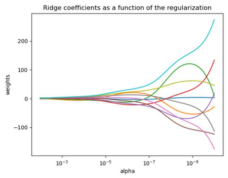
[Compressive sensing: tomography reconstruction with L1 prior (Lasso)]



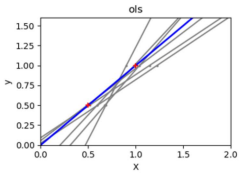
[Prediction Latency]



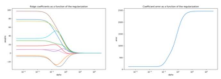
[Comparison of kernel ridge and Gaussian process regression]



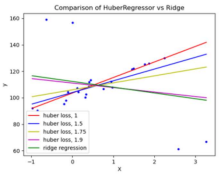
[Plot Ridge coefficients as a function of the regularization]



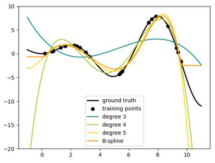
[Ordinary Least Squares and Ridge Regression Variance]



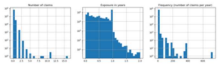
[Plot Ridge coefficients as a function of the L2 regularization]



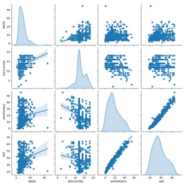
[HuberRegressor vs Ridge on dataset with strong outliers]



[Polynomial and Spline interpolation]



[Poisson regression and non-normal loss]



[Common pitfalls in the interpretation of coefficients of linear models]