Aryaman Gautam

J001

SKLEARN API

LINEAR REGRESSION:

Linear Regression 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.

It's parameters are:

- fit_intercept bool, default=True

  Whether to calculate the intercept for this model. If False then no intercept is used for calculation.

- 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 for n_targets > 1 and sufficient large problems.

- positive bool, default=False

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

LinearRegression(*, fit_intercept=True, normalize=False, copy_X=True, n_jobs=None, positive=False)

Attribute:

coef_array of shape (n_features,) or (n_targets, n_features)

Estimated coefficients for the linear regression problem.

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.

Methods

fit(X, y[, sample_weight])	Fit linear 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

LOGISTIC REGRESSION:

It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable.

Fit(X,y)-fit the model according to the given training data

Predict(x)-predict class labels

Score(X,y)-returns mean accuracy on the given test data and label

Code:

sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)

RIDGE:

||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)).

Ridge regression penalizes the model based on the sum of squares of magnitude of the coefficients.

Alpha-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.

It’s parameters are:

- alpha{float, ndarray of shape (n_targets,)}, default=1.0

  Regularization strength; must be a positive float. Larger values specify stronger regularization

- fit_intercept: bool, default=True

  Whether to fit the intercept for this model

- normalize: bool, default=False

  This parameter is ignored when fit_intercept is set to False.

- copy_Xbool, default=True

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

- max_iterint, default=None

  Maximum number of iterations for conjugate gradient solver

- tolfloat, default=1e-3

  Precision of the solution.

- solver{‘auto’, ‘svd’, ‘cholesky’, ‘lsqr’, ‘sparse_cg’, ‘sag’, ‘saga’}, default=’auto’

  Solver to use in the computational routines. ‘auto’ chooses the solver automatically based on the type of data.

- random_stateint, RandomState instance, default=None

  Used when solver == ‘sag’ or ‘saga’ to shuffle the data

Ridge(alpha=1.0, *, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)

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.

LASSO :

LASSO regression penalizes the model based on the sum of magnitude of the coefficients.

Technically the Lasso model is optimizing the same objective function as the Elastic Net with l1_ratio=1.0 (no L2 penalty).

Lasso(alpha=1.0, *, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')

Parameters:

- alpha float, default=1.0

Constant that multiplies the L1 term. Defaults to 1.0. alpha = 0 is equivalent to an ordinary least square, solved by the LinearRegression object. For numerical reasons, using alpha = 0 with the Lasso object is not advised

- 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.

- Precompute bool or array-like of shape (n_features, n_features), default=False

Whether to use a precomputed Gram matrix to speed up calculations. The Gram matrix can also be passed as argument. For sparse input this option is always False to preserve sparsity.

- copy_X bool, default=True

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

- max_iter int, default=1000

The maximum number of iterations.

- Tol float, default=1e-4

The tolerance for the optimization: if the updates are smaller than tol, the optimization code checks the dual gap for optimality and continues until it is smaller than tol.

- warm_start bool, default=False

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erases the previous solution..

- Positive bool, default=False

When set to True, forces the coefficients to be positive.

- random_stateint, RandomState instance, default=None

The seed of the pseudo random number generator that selects a random feature to update. Used when selection == ‘random’.

Pass an int for reproducible output across multiple function calls.

- selection{‘cyclic’, ‘random’}, default=’cyclic’

If set to ‘random’, a random coefficient is updated every iteration rather than looping over features sequentially by default. This (setting to ‘random’) often leads to significantly faster convergence especially when tol is higher than 1e-4