API Reference

This is the class and function reference of scikit-learn. Please refer to the full user guide for further details, as the class and function raw specifications may not be enough to give full guidelines on their uses. For reference on concepts repeated across the API, see Glossary of Common Terms and API Elements.

sklearn.base: Base classes and utility functions

Base classes for all estimators.

Base classes

base.BaseEstimator	Base class for all estimators in scikit-learn.
base.BiclusterMixin	Mixin class for all bicluster estimators in scikit-learn.
base.ClassifierMixin	Mixin class for all classifiers in scikit-learn.
base.ClusterMixin	Mixin class for all cluster estimators in scikit-learn.
base.DensityMixin	Mixin class for all density estimators in scikit-learn.
base.RegressorMixin	Mixin class for all regression estimators in scikit-learn.
base.TransformerMixin	Mixin class for all transformers in scikit-learn.
feature_selection.SelectorMixin	Transformer mixin that performs feature selection given a support mask

Functions

base.clone(estimator, *[, safe])	Constructs a new unfitted estimator with the same parameters.
base.is_classifier(estimator)	Return True if the given estimator is (probably) a classifier.
base.is_regressor(estimator)	Return True if the given estimator is (probably) a regressor.
config_context(**new_config)	Context manager for global scikit-learn configuration
get_config()	Retrieve current values for configuration set by set_config
set_config([assume_finite, working_memory, ...])	Set global scikit-learn configuration
show_versions()	Print useful debugging information"

sklearn.calibration: Probability Calibration

Calibration of predicted probabilities.

User guide: See the Probability calibration section for further details.

calibration.CalibratedClassifierCV([...])	Probability calibration with isotonic regression or logistic regression.

calibration.calibration_curve(y_true, y_prob, *)	Compute true and predicted probabilities for a calibration curve.

sklearn.cluster: Clustering

The sklearn.cluster module gathers popular unsupervised clustering algorithms.

User guide: See the Clustering and Biclustering sections for further details.

Classes

cluster.AffinityPropagation(*[, damping, ...])	Perform Affinity Propagation Clustering of data.
cluster.AgglomerativeClustering([...])	Agglomerative Clustering
cluster.Birch(*[, threshold, ...])	Implements the BIRCH clustering algorithm.
cluster.DBSCAN([eps, min_samples, metric, ...])	Perform DBSCAN clustering from vector array or distance matrix.
cluster.FeatureAgglomeration([n_clusters, ...])	Agglomerate features.
cluster.KMeans([n_clusters, init, n_init, ...])	K-Means clustering.
cluster.MiniBatchKMeans([n_clusters, init, ...])	Mini-Batch K-Means clustering.
cluster.MeanShift(*[, bandwidth, seeds, ...])	Mean shift clustering using a flat kernel.
cluster.OPTICS(*[, min_samples, max_eps, ...])	Estimate clustering structure from vector array.
cluster.SpectralClustering([n_clusters, ...])	Apply clustering to a projection of the normalized Laplacian.
cluster.SpectralBiclustering([n_clusters, ...])	Spectral biclustering (Kluger, 2003).
cluster.SpectralCoclustering([n_clusters, ...])	Spectral Co-Clustering algorithm (Dhillon, 2001).

Functions

Toggle Menu

cluster.affinity_propagation(S, *[, ...])	Perform Affinity Propagation Clustering of data.
cluster.cluster_optics_dbscan(*, ...)	Performs DBSCAN extraction for an arbitrary epsilon.
cluster.cluster_optics_xi(*, reachability, ...)	Automatically extract clusters according to the Xi-steep method.
cluster.compute_optics_graph(X, *, ...)	Computes the OPTICS reachability graph.
cluster.dbscan(X[, eps, min_samples, ...])	Perform DBSCAN clustering from vector array or distance matrix.
cluster.estimate_bandwidth(X, *[, quantile, ...])	Estimate the bandwidth to use with the mean-shift algorithm.
cluster.k_means(X, n_clusters, *[, ...])	K-means clustering algorithm.
cluster.kmeans_plusplus(X, n_clusters, *[, ...])	Init n_clusters seeds according to k-means++
cluster.mean_shift(X, *[, bandwidth, seeds, ...])	Perform mean shift clustering of data using a flat kernel.
cluster.spectral_clustering(affinity, *[, ...])	Apply clustering to a projection of the normalized Laplacian.
cluster.ward_tree(X, *[, connectivity, ...])	Ward clustering based on a Feature matrix.

sklearn.compose: Composite Estimators

Meta-estimators for building composite models with transformers

In addition to its current contents, this module will eventually be home to refurbished versions of Pipeline and FeatureUnion.

User guide: See the Pipelines and composite estimators section for further details.

compose.ColumnTransformer(transformers, *[, ...])	Applies transformers to columns of an array or pandas DataFrame.
compose.TransformedTargetRegressor([...])	Meta-estimator to regress on a transformed target.

compose.make_column_transformer(*transformers)	Construct a ColumnTransformer from the given transformers.
compose.make_column_selector([pattern, ...])	Create a callable to select columns to be used with `ColumnTransformer`.

sklearn.covariance: Covariance Estimators

The `sklearn.covariance` module includes methods and algorithms to robustly estimate the covariance of features given a set of points. The precision matrix defined as the inverse of the covariance is also estimated. Covariance estimation is closely related to the theory of Gaussian Graphical Models.

User guide: See the Covariance estimation section for further details.

covariance.EmpiricalCovariance(*[, ...])	Maximum likelihood covariance estimator
covariance.EllipticEnvelope(*[, ...])	An object for detecting outliers in a Gaussian distributed dataset.
covariance.GraphicalLasso([alpha, mode, ...])	Sparse inverse covariance estimation with an l1-penalized estimator.
covariance.GraphicalLassoCV(*[, alphas, ...])	Sparse inverse covariance w/ cross-validated choice of the l1 penalty.
covariance.LedoitWolf(*[, store_precision, ...])	LedoitWolf Estimator
covariance.MinCovDet(*[, store_precision, ...])	Minimum Covariance Determinant (MCD): robust estimator of covariance.
covariance.OAS(*[, store_precision, ...])	Oracle Approximating Shrinkage Estimator
covariance.ShrunkCovariance(*[, ...])	Covariance estimator with shrinkage

covariance.empirical_covariance(X, *[, ...])	Computes the Maximum likelihood covariance estimator
covariance.graphical_lasso(emp_cov, alpha, *)	l1-penalized covariance estimator
covariance.ledoit_wolf(X, *[, ...])	Estimates the shrunk Ledoit-Wolf covariance matrix.
covariance.oas(X, *[, assume_centered])	Estimate covariance with the Oracle Approximating Shrinkage algorithm.
covariance.shrunk_covariance(emp_cov[, ...])	Calculates a covariance matrix shrunk on the diagonal

sklearn.cross_decomposition: Cross decomposition

User guide: See the Cross decomposition section for further details.

cross_decomposition.CCA([n_components, ...])	Canonical Correlation Analysis, also known as “Mode B” PLS.
cross_decomposition.PLSCanonical([...])	Partial Least Squares transformer and regressor.
cross_decomposition.PLSRegression([...])	PLS regression
cross_decomposition.PLSSVD([n_components, ...])	Partial Least Square SVD.

sklearn.datasets: Datasets

The `sklearn.datasets` module includes utilities to load datasets, including methods to load and fetch popular reference datasets. It also

Toggle Menu

artificial data generators.

User guide: See the Dataset loading utilities section for further details.

Loaders

`datasets.clear_data_home`([data_home])	Delete all the content of the data home cache.
`datasets.dump_svmlight_file`(X, y, f, *[, ...])	Dump the dataset in svmlight / libsvm file format.
`datasets.fetch_20newsgroups`(*[, data_home, ...])	Load the filenames and data from the 20 newsgroups dataset (classification).
`datasets.fetch_20newsgroups_vectorized`(*[, ...])	Load and vectorize the 20 newsgroups dataset (classification).
`datasets.fetch_california_housing`(*[, ...])	Load the California housing dataset (regression).
`datasets.fetch_covtype`(*[, data_home, ...])	Load the covertype dataset (classification).
`datasets.fetch_kddcup99`(*[, subset, ...])	Load the kddcup99 dataset (classification).
`datasets.fetch_lfw_pairs`(*[, subset, ...])	Load the Labeled Faces in the Wild (LFW) pairs dataset (classification).
`datasets.fetch_lfw_people`(*[, data_home, ...])	Load the Labeled Faces in the Wild (LFW) people dataset (classification).
`datasets.fetch_olivetti_faces`(*[, ...])	Load the Olivetti faces data-set from AT&T (classification).
`datasets.fetch_openml`([name, version, ...])	Fetch dataset from openml by name or dataset id.
`datasets.fetch_rcv1`(*[, data_home, subset, ...])	Load the RCV1 multilabel dataset (classification).
`datasets.fetch_species_distributions`(*[, ...])	Loader for species distribution dataset from Phillips et.
`datasets.get_data_home`([data_home])	Return the path of the scikit-learn data dir.
`datasets.load_boston`(*[, return_X_y])	Load and return the boston house-prices dataset (regression).
`datasets.load_breast_cancer`(*[, return_X_y, ...])	Load and return the breast cancer wisconsin dataset (classification).
`datasets.load_diabetes`(*[, return_X_y, as_frame])	Load and return the diabetes dataset (regression).
`datasets.load_digits`(*[, n_class, ...])	Load and return the digits dataset (classification).
`datasets.load_files`(container_path, *[, ...])	Load text files with categories as subfolder names.
`datasets.load_iris`(*[, return_X_y, as_frame])	Load and return the iris dataset (classification).
`datasets.load_linnerud`(*[, return_X_y, as_frame])	Load and return the physical excercise linnerud dataset.
`datasets.load_sample_image`(image_name)	Load the numpy array of a single sample image
`datasets.load_sample_images`()	Load sample images for image manipulation.
`datasets.load_svmlight_file`(f, *[, ...])	Load datasets in the svmlight / libsvm format into sparse CSR matrix
`datasets.load_svmlight_files`(files, *[, ...])	Load dataset from multiple files in SVMlight format
`datasets.load_wine`(*[, return_X_y, as_frame])	Load and return the wine dataset (classification).

Samples generator

`datasets.make_biclusters`(shape, n_clusters, *)	Generate an array with constant block diagonal structure for biclustering.
`datasets.make_blobs`([n_samples, n_features, ...])	Generate isotropic Gaussian blobs for clustering.
`datasets.make_checkerboard`(shape, n_clusters, *)	Generate an array with block checkerboard structure for biclustering.
`datasets.make_circles`([n_samples, shuffle, ...])	Make a large circle containing a smaller circle in 2d.
`datasets.make_classification`([n_samples, ...])	Generate a random n-class classification problem.
`datasets.make_friedman1`([n_samples, ...])	Generate the “Friedman #1” regression problem.
`datasets.make_friedman2`([n_samples, noise, ...])	Generate the “Friedman #2” regression problem.
`datasets.make_friedman3`([n_samples, noise, ...])	Generate the “Friedman #3” regression problem.
`datasets.make_gaussian_quantiles`(*[, mean, ...])	Generate isotropic Gaussian and label samples by quantile.
`datasets.make_hastie_10_2`([n_samples, ...])	Generates data for binary classification used in Hastie et al.
`datasets.make_low_rank_matrix`([n_samples, ...])	Generate a mostly low rank matrix with bell-shaped singular values.
`datasets.make_moons`([n_samples, shuffle, ...])	Make two interleaving half circles.
`datasets.make_multilabel_classification`([...])	Generate a random multilabel classification problem.
`datasets.make_regression`([n_samples, ...])	Generate a random regression problem.
`datasets.make_s_curve`([n_samples, noise, ...])	Generate an S curve dataset.
`datasets.make_sparse_coded_signal`(n_samples, ...)	Generate a signal as a sparse combination of dictionary elements.
`datasets.make_sparse_spd_matrix`([dim, ...])	Generate a sparse symmetric definite positive matrix.
`datasets.make_sparse_uncorrelated`([...])	Generate a random regression problem with sparse uncorrelated design.
`datasets.make_spd_matrix`(n_dim, *[, ...])	Generate a random symmetric, positive-definite matrix.
`datasets.make_swiss_roll`([n_samples, noise, ...])	Generate a swiss roll dataset.

`sklearn.decomposition`: Matrix Decomposition

The `sklearn.decomposition` module includes matrix decomposition algorithms, including among others PCA, NMF or ICA. Most of the algo- module can be regarded as dimensionality reduction techniques.

Toggle Menu

User guide: See the Decomposing signals in components (matrix factorization problems) section for further details.

decomposition.DictionaryLearning([...])	Dictionary learning
decomposition.FactorAnalysis([n_components, ...])	Factor Analysis (FA).
decomposition.FastICA([n_components, ...])	FastICA: a fast algorithm for Independent Component Analysis.
decomposition.IncrementalPCA([n_components, ...])	Incremental principal components analysis (IPCA).
decomposition.KernelPCA([n_components, ...])	Kernel Principal component analysis (KPCA).
decomposition.LatentDirichletAllocation([...])	Latent Dirichlet Allocation with online variational Bayes algorithm
decomposition.MiniBatchDictionaryLearning([...])	Mini-batch dictionary learning
decomposition.MiniBatchSparsePCA([...])	Mini-batch Sparse Principal Components Analysis
decomposition.NMF([n_components, init, ...])	Non-Negative Matrix Factorization (NMF).
decomposition.PCA([n_components, copy, ...])	Principal component analysis (PCA).
decomposition.SparsePCA([n_components, ...])	Sparse Principal Components Analysis (SparsePCA).
decomposition.SparseCoder(dictionary, *[, ...])	Sparse coding
decomposition.TruncatedSVD([n_components, ...])	Dimensionality reduction using truncated SVD (aka LSA).

decomposition.dict_learning(X, n_components, ...)	Solves a dictionary learning matrix factorization problem.
decomposition.dict_learning_online(X[, ...])	Solves a dictionary learning matrix factorization problem online.
decomposition.fastica(X[, n_components, ...])	Perform Fast Independent Component Analysis.
decomposition.non_negative_factorization(X)	Compute Non-negative Matrix Factorization (NMF).
decomposition.sparse_encode(X, dictionary, *)	Sparse coding

sklearn.discriminant_analysis: Discriminant Analysis

Linear Discriminant Analysis and Quadratic Discriminant Analysis

User guide: See the Linear and Quadratic Discriminant Analysis section for further details.

discriminant_analysis.LinearDiscriminantAnalysis([...])	Linear Discriminant Analysis
discriminant_analysis.QuadraticDiscriminantAnalysis(*)	Quadratic Discriminant Analysis

sklearn.dummy: Dummy estimators

User guide: See the Metrics and scoring: quantifying the quality of predictions section for further details.

dummy.DummyClassifier(*[, strategy, ...])	DummyClassifier is a classifier that makes predictions using simple rules.
dummy.DummyRegressor(*[, strategy, ...])	DummyRegressor is a regressor that makes predictions using simple rules.

sklearn.ensemble: Ensemble Methods ¶

The sklearn.ensemble module includes ensemble-based methods for classification, regression and anomaly detection.

User guide: See the Ensemble methods section for further details.

ensemble.AdaBoostClassifier([...])	An AdaBoost classifier.
ensemble.AdaBoostRegressor([base_estimator, ...])	An AdaBoost regressor.
ensemble.BaggingClassifier([base_estimator, ...])	A Bagging classifier.
ensemble.BaggingRegressor([base_estimator, ...])	A Bagging regressor.
ensemble.ExtraTreesClassifier([...])	An extra-trees classifier.
ensemble.ExtraTreesRegressor([n_estimators, ...])	An extra-trees regressor.
ensemble.GradientBoostingClassifier(*[, ...])	Gradient Boosting for classification.
ensemble.GradientBoostingRegressor(*[, ...])	Gradient Boosting for regression.
ensemble.IsolationForest(*[, n_estimators, ...])	Isolation Forest Algorithm.
ensemble.RandomForestClassifier([...])	A random forest classifier.
ensemble.RandomForestRegressor([...])	A random forest regressor.
ensemble.RandomTreesEmbedding([...])	An ensemble of totally random trees.
ensemble.StackingClassifier(estimators[, ...])	Stack of estimators with a final classifier.
ensemble.StackingRegressor(estimators[, ...])	Stack of estimators with a final regressor.
ensemble.VotingClassifier(estimators, *[, ...])	Soft Voting/Majority Rule classifier for unfitted estimators.
ensemble.VotingRegressor(estimators, *[, ...])	Prediction voting regressor for unfitted estimators.
ensemble.HistGradientBoostingRegressor([...])	Histogram-based Gradient Boosting Regression Tree.
ensemble.HistGradientBoostingClassifier([...])	Histogram-based Gradient Boosting Classification Tree.

Toggle Menu

sklearn.exceptions: Exceptions and warnings

The sklearn.exceptions module includes all custom warnings and error classes used across scikit-learn.

exceptions.ConvergenceWarning	Custom warning to capture convergence problems
exceptions.DataConversionWarning	Warning used to notify implicit data conversions happening in the code.
exceptions.DataDimensionalityWarning	Custom warning to notify potential issues with data dimensionality.
exceptions.EfficiencyWarning	Warning used to notify the user of inefficient computation.
exceptions.FitFailedWarning	Warning class used if there is an error while fitting the estimator.
exceptions.NotFittedError	Exception class to raise if estimator is used before fitting.
exceptions.UndefinedMetricWarning	Warning used when the metric is invalid

sklearn.experimental: Experimental

The sklearn.experimental module provides importable modules that enable the use of experimental features or estimators.

The features and estimators that are experimental aren't subject to deprecation cycles. Use them at your own risks!

experimental.enable_hist_gradient_boosting	Enables histogram-based gradient boosting estimators.
experimental.enable_iterative_imputer	Enables IterativeImputer
experimental.enable_halving_search_cv	Enables Successive Halving search-estimators

sklearn.feature_extraction: Feature Extraction

The sklearn.feature_extraction module deals with feature extraction from raw data. It currently includes methods to extract features from text and images.

User guide: See the Feature extraction section for further details.

feature_extraction.DictVectorizer(*[, ...])	Transforms lists of feature-value mappings to vectors.
feature_extraction.FeatureHasher([...])	Implements feature hashing, aka the hashing trick.

From images

The sklearn.feature_extraction.image submodule gathers utilities to extract features from images.

feature_extraction.image.extract_patches_2d(...)	Reshape a 2D image into a collection of patches
feature_extraction.image.grid_to_graph(n_x, n_y)	Graph of the pixel-to-pixel connections
feature_extraction.image.img_to_graph(img, *)	Graph of the pixel-to-pixel gradient connections
feature_extraction.image.reconstruct_from_patches_2d(...)	Reconstruct the image from all of its patches.
feature_extraction.image.PatchExtractor(*[, ...])	Extracts patches from a collection of images

From text

The sklearn.feature_extraction.text submodule gathers utilities to build feature vectors from text documents.

feature_extraction.text.CountVectorizer(*[, ...])	Convert a collection of text documents to a matrix of token counts
feature_extraction.text.HashingVectorizer(*)	Convert a collection of text documents to a matrix of token occurrences
feature_extraction.text.TfidfTransformer(*)	Transform a count matrix to a normalized tf or tf-idf representation
feature_extraction.text.TfidfVectorizer(*[, ...])	Convert a collection of raw documents to a matrix of TF-IDF features.

sklearn.feature_selection: Feature Selection

The sklearn.feature_selection module implements feature selection algorithms. It currently includes univariate filter selection methods and the recursive feature elimination algorithm.

User guide: See the Feature selection section for further details.

feature_selection.GenericUnivariateSelect([...])	Univariate feature selector with configurable strategy.
feature_selection.SelectPercentile([...])	Select features according to a percentile of the highest scores.
feature_selection.SelectKBest([score_func, k])	Select features according to the k highest scores.
feature_selection.SelectFpr([score_func, alpha])	Filter: Select the pvalues below alpha based on a FPR test.
feature_selection.SelectFdr([score_func, alpha])	Filter: Select the p-values for an estimated false discovery rate
feature_selection.SelectFromModel(estimator, *)	Meta-transformer for selecting features based on importance weights.

Toggle Menu

feature_selection.SelectFwe([score_func, alpha])	Filter: Select the p-values corresponding to Family-wise error rate
feature_selection.SequentialFeatureSelector(...)	Transformer that performs Sequential Feature Selection.
feature_selection.RFE(estimator, *[, ...])	Feature ranking with recursive feature elimination.
feature_selection.RFECV(estimator, *[, ...])	Feature ranking with recursive feature elimination and cross-validated selection of the best number of features.
feature_selection.VarianceThreshold([threshold])	Feature selector that removes all low-variance features.

feature_selection.chi2(X, y)	Compute chi-squared stats between each non-negative feature and class.
feature_selection.f_classif(X, y)	Compute the ANOVA F-value for the provided sample.
feature_selection.f_regression(X, y, *[, center])	Univariate linear regression tests.
feature_selection.mutual_info_classif(X, y, *)	Estimate mutual information for a discrete target variable.
feature_selection.mutual_info_regression(X, y, *)	Estimate mutual information for a continuous target variable.

sklearn.gaussian_process: Gaussian Processes

The sklearn.gaussian_process module implements Gaussian Process based regression and classification.

User guide: See the Gaussian Processes section for further details.

gaussian_process.GaussianProcessClassifier([...])	Gaussian process classification (GPC) based on Laplace approximation.
gaussian_process.GaussianProcessRegressor([...])	Gaussian process regression (GPR).

Kernels:

gaussian_process.kernels.CompoundKernel(kernels)	Kernel which is composed of a set of other kernels.
gaussian_process.kernels.ConstantKernel([...])	Constant kernel.
gaussian_process.kernels.DotProduct([...])	Dot-Product kernel.
gaussian_process.kernels.ExpSineSquared([...])	Exp-Sine-Squared kernel (aka periodic kernel).
gaussian_process.kernels.Exponentiation(...)	The Exponentiation kernel takes one base kernel and a scalar parameter p and combines them via
gaussian_process.kernels.Hyperparameter(...)	A kernel hyperparameter's specification in form of a namedtuple.
gaussian_process.kernels.Kernel()	Base class for all kernels.
gaussian_process.kernels.Matern([...])	Matern kernel.
gaussian_process.kernels.PairwiseKernel([...])	Wrapper for kernels in sklearn.metrics.pairwise.
gaussian_process.kernels.Product(k1, k2)	The `Product` kernel takes two kernels k_1 and k_2 and combines them via
gaussian_process.kernels.RBF([length_scale, ...])	Radial-basis function kernel (aka squared-exponential kernel).
gaussian_process.kernels.RationalQuadratic([...])	Rational Quadratic kernel.
gaussian_process.kernels.Sum(k1, k2)	The `Sum` kernel takes two kernels k_1 and k_2 and combines them via
gaussian_process.kernels.WhiteKernel([...])	White kernel.

sklearn.impute: Impute

Transformers for missing value imputation

User guide: See the Imputation of missing values section for further details.

impute.SimpleImputer(*[, missing_values, ...])	Imputation transformer for completing missing values.
impute.IterativeImputer([estimator, ...])	Multivariate imputer that estimates each feature from all the others.
impute.MissingIndicator(*[, missing_values, ...])	Binary indicators for missing values.
impute.KNNImputer(*[, missing_values, ...])	Imputation for completing missing values using k-Nearest Neighbors.

sklearn.inspection: Inspection

The sklearn.inspection module includes tools for model inspection.

inspection.partial_dependence(estimator, X, ...)	Partial dependence of `features`.
inspection.permutation_importance(estimator, ...)	Permutation importance for feature evaluation [Rd9e56ef97513-BRE].

Plotting

inspection.PartialDependenceDisplay(...[, ...])	Partial Dependence Plot (PDP).
inspection.plot_partial_dependence(...[, ...])	Partial dependence (PD) and individual conditional expectation (ICE) plots.

Toggle Menu

sklearn.isotonic: Isotonic regression

User guide: See the Isotonic regression section for further details.

isotonic.IsotonicRegression(*[, y_min, ...])	Isotonic regression model.

isotonic.check_increasing(x, y)	Determine whether y is monotonically correlated with x.
isotonic.isotonic_regression(y, *[, ...])	Solve the isotonic regression model.

sklearn.kernel_approximation: Kernel Approximation

The sklearn.kernel_approximation module implements several approximate kernel feature maps based on Fourier transforms and Count Sketches.

User guide: See the Kernel Approximation section for further details.

kernel_approximation.AdditiveChi2Sampler(*)	Approximate feature map for additive chi2 kernel.
kernel_approximation.Nystroem([kernel, ...])	Approximate a kernel map using a subset of the training data.
kernel_approximation.PolynomialCountSketch(*)	Polynomial kernel approximation via Tensor Sketch.
kernel_approximation.RBFSampler(*[, gamma, ...])	Approximates feature map of an RBF kernel by Monte Carlo approximation of its Fourier transform.
kernel_approximation.SkewedChi2Sampler(*[, ...])	Approximates feature map of the "skewed chi-squared" kernel by Monte Carlo approximation of its Fourier transform.

sklearn.kernel_ridge: Kernel Ridge Regression

Module sklearn.kernel_ridge implements kernel ridge regression.

User guide: See the Kernel ridge regression section for further details.

kernel_ridge.KernelRidge([alpha, kernel, ...])	Kernel ridge regression.

sklearn.linear_model: Linear Models

The sklearn.linear_model module implements a variety of linear models.

User guide: See the Linear Models section for further details.

The following subsections are only rough guidelines: the same estimator can fall into multiple categories, depending on its parameters.

Linear classifiers

linear_model.LogisticRegression([penalty, ...])	Logistic Regression (aka logit, MaxEnt) classifier.
linear_model.LogisticRegressionCV(*[, Cs, ...])	Logistic Regression CV (aka logit, MaxEnt) classifier.
linear_model.PassiveAggressiveClassifier(*)	Passive Aggressive Classifier
linear_model.Perceptron(*[, penalty, alpha, ...])	Read more in the User Guide.
linear_model.RidgeClassifier([alpha, ...])	Classifier using Ridge regression.
linear_model.RidgeClassifierCV([alphas, ...])	Ridge classifier with built-in cross-validation.
linear_model.SGDClassifier([loss, penalty, ...])	Linear classifiers (SVM, logistic regression, etc.) with SGD training.

Classical linear regressors

linear_model.LinearRegression(*[, ...])	Ordinary least squares Linear Regression.
linear_model.Ridge([alpha, fit_intercept, ...])	Linear least squares with l2 regularization.
linear_model.RidgeCV([alphas, ...])	Ridge regression with built-in cross-validation.
linear_model.SGDRegressor([loss, penalty, ...])	Linear model fitted by minimizing a regularized empirical loss with SGD

Regressors with variable selection

The following estimators have built-in variable selection fitting procedures, but any estimator using a L1 or elastic-net penalty also performs variable selection: typically SGDRegressor or SGDClassifier with an appropriate penalty.

linear_model.ElasticNet([alpha, l1_ratio, ...])	Linear regression with combined L1 and L2 priors as regularizer.
linear_model.ElasticNetCV(*[, l1_ratio, ...])	Elastic Net model with iterative fitting along a regularization path.
linear_model.Lars(*[, fit_intercept, ...])	Least Angle Regression model a.k.a.
.LarsCV(*[, fit_intercept, ...])	Cross-validated Least Angle Regression model.

Toggle Menu

linear_model.Lasso([alpha, fit_intercept, ...])	Linear Model trained with L1 prior as regularizer (aka the Lasso)
linear_model.LassoCV(*[, eps, n_alphas, ...])	Lasso linear model with iterative fitting along a regularization path.
linear_model.LassoLars([alpha, ...])	Lasso model fit with Least Angle Regression a.k.a.
linear_model.LassoLarsCV(*[, fit_intercept, ...])	Cross-validated Lasso, using the LARS algorithm.
linear_model.LassoLarsIC([criterion, ...])	Lasso model fit with Lars using BIC or AIC for model selection
linear_model.OrthogonalMatchingPursuit(*[, ...])	Orthogonal Matching Pursuit model (OMP).
linear_model.OrthogonalMatchingPursuitCV(*)	Cross-validated Orthogonal Matching Pursuit model (OMP).

Bayesian regressors

linear_model.ARDRegression(*[, n_iter, tol, ...])	Bayesian ARD regression.
linear_model.BayesianRidge(*[, n_iter, tol, ...])	Bayesian ridge regression.

Multi-task linear regressors with variable selection

These estimators fit multiple regression problems (or tasks) jointly, while inducing sparse coefficients. While the inferred coefficients may differ between the tasks, they are constrained to agree on the features that are selected (non-zero coefficients).

linear_model.MultiTaskElasticNet([alpha, ...])	Multi-task ElasticNet model trained with L1/L2 mixed-norm as regularizer.
linear_model.MultiTaskElasticNetCV(*[, ...])	Multi-task L1/L2 ElasticNet with built-in cross-validation.
linear_model.MultiTaskLasso([alpha, ...])	Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer.
linear_model.MultiTaskLassoCV(*[, eps, ...])	Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer.

Outlier-robust regressors

Any estimator using the Huber loss would also be robust to outliers, e.g. `SGDRegressor` with `loss='huber'`.

linear_model.HuberRegressor(*[, epsilon, ...])	Linear regression model that is robust to outliers.
linear_model.RANSACRegressor([...])	RANSAC (RANdom SAmple Consensus) algorithm.
linear_model.TheilSenRegressor(*[, ...])	Theil-Sen Estimator: robust multivariate regression model.

Generalized linear models (GLM) for regression

These models allow for response variables to have error distributions other than a normal distribution:

linear_model.PoissonRegressor(*[, alpha, ...])	Generalized Linear Model with a Poisson distribution.
linear_model.TweedieRegressor(*[, power, ...])	Generalized Linear Model with a Tweedie distribution.
linear_model.GammaRegressor(*[, alpha, ...])	Generalized Linear Model with a Gamma distribution.

Miscellaneous

linear_model.PassiveAggressiveRegressor(*[, ...])	Passive Aggressive Regressor
linear_model.enet_path(X, y, *[, l1_ratio, ...])	Compute elastic net path with coordinate descent.
linear_model.lars_path(X, y[, Xy, Gram, ...])	Compute Least Angle Regression or Lasso path using LARS algorithm [1]
linear_model.lars_path_gram(Xy, Gram, *, ...)	lars_path in the sufficient stats mode [1]
linear_model.lasso_path(X, y, *[, eps, ...])	Compute Lasso path with coordinate descent
linear_model.orthogonal_mp(X, y, *[, ...])	Orthogonal Matching Pursuit (OMP).
linear_model.orthogonal_mp_gram(Gram, Xy, *)	Gram Orthogonal Matching Pursuit (OMP).
linear_model.ridge_regression(X, y, alpha, *)	Solve the ridge equation by the method of normal equations.

sklearn.manifold: Manifold Learning

The `sklearn.manifold` module implements data embedding techniques.

User guide: See the Manifold learning section for further details.

manifold.Isomap(*[, n_neighbors, ...])	Isomap Embedding
manifold.LocallyLinearEmbedding(*[, ...])	Locally Linear Embedding
manifold.MDS([n_components, metric, n_init, ...])	Multidimensional scaling.
manifold.SpectralEmbedding([n_components, ...])	Spectral embedding for non-linear dimensionality reduction.
manifold.TSNE([n_components, perplexity, ...])	t-distributed Stochastic Neighbor Embedding.

manifold.locally_linear_embedding(X, *, ...)	Perform a Locally Linear Embedding analysis on the data.
manifold.smacof(dissimilarities, *[, ...])	Computes multidimensional scaling using the SMACOF algorithm.
manifold.spectral_embedding(adjacency, *[, ...])	Project the sample on the first eigenvectors of the graph Laplacian.
manifold.trustworthiness(X, X_embedded, *[, ...])	Expresses to what extent the local structure is retained.

sklearn.metrics: Metrics

See the Metrics and scoring: quantifying the quality of predictions section and the Pairwise metrics, Affinities and Kernels section of the user guide for further details.

The sklearn.metrics module includes score functions, performance metrics and pairwise metrics and distance computations.

Model Selection Interface

See the The scoring parameter: defining model evaluation rules section of the user guide for further details.

metrics.check_scoring(estimator[, scoring, ...])	Determine scorer from user options.
metrics.get_scorer(scoring)	Get a scorer from string.
metrics.make_scorer(score_func, *[, ...])	Make a scorer from a performance metric or loss function.

Classification metrics

See the Classification metrics section of the user guide for further details.

metrics.accuracy_score(y_true, y_pred, *[, ...])	Accuracy classification score.
metrics.auc(x, y)	Compute Area Under the Curve (AUC) using the trapezoidal rule.
metrics.average_precision_score(y_true, ...)	Compute average precision (AP) from prediction scores.
metrics.balanced_accuracy_score(y_true, ...)	Compute the balanced accuracy.
metrics.brier_score_loss(y_true, y_prob, *)	Compute the Brier score loss.
metrics.classification_report(y_true, y_pred, *)	Build a text report showing the main classification metrics.
metrics.cohen_kappa_score(y1, y2, *[, ...])	Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix(y_true, y_pred, *)	Compute confusion matrix to evaluate the accuracy of a classification.
metrics.dcg_score(y_true, y_score, *[, k, ...])	Compute Discounted Cumulative Gain.
metrics.det_curve(y_true, y_score[, ...])	Compute error rates for different probability thresholds.
metrics.f1_score(y_true, y_pred, *[, ...])	Compute the F1 score, also known as balanced F-score or F-measure.
metrics.fbeta_score(y_true, y_pred, *, beta)	Compute the F-beta score.
metrics.hamming_loss(y_true, y_pred, *[, ...])	Compute the average Hamming loss.
metrics.hinge_loss(y_true, pred_decision, *)	Average hinge loss (non-regularized).
metrics.jaccard_score(y_true, y_pred, *[, ...])	Jaccard similarity coefficient score.
metrics.log_loss(y_true, y_pred, *[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef(y_true, y_pred, *)	Compute the Matthews correlation coefficient (MCC).
metrics.multilabel_confusion_matrix(y_true, ...)	Compute a confusion matrix for each class or sample.
metrics.ndcg_score(y_true, y_score, *[, k, ...])	Compute Normalized Discounted Cumulative Gain.
metrics.precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds.
metrics.precision_recall_fscore_support(...)	Compute precision, recall, F-measure and support for each class.
metrics.precision_score(y_true, y_pred, *[, ...])	Compute the precision.
metrics.recall_score(y_true, y_pred, *[, ...])	Compute the recall.
metrics.roc_auc_score(y_true, y_score, *[, ...])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
metrics.roc_curve(y_true, y_score, *[, ...])	Compute Receiver operating characteristic (ROC).
metrics.top_k_accuracy_score(y_true, y_score, *)	Top-k Accuracy classification score.
metrics.zero_one_loss(y_true, y_pred, *[, ...])	Zero-one classification loss.

Regression metrics

See the Regression metrics section of the user guide for further details.

metrics.explained_variance_score(y_true, ...)	Explained variance regression score function.
metrics.max_error(y_true, y_pred)	max_error metric calculates the maximum residual error.
metrics.mean_absolute_error(y_true, y_pred, *)	Mean absolute error regression loss.
metrics.mean_squared_error(y_true, y_pred, *)	Mean squared error regression loss.
metrics.mean_squared_log_error(y_true, y_pred, *)	Mean squared logarithmic error regression loss.
metrics.median_absolute_error(y_true, y_pred, *)	Median absolute error regression loss.
metrics.mean_absolute_percentage_error(...)	Mean absolute percentage error regression loss.
metrics.r2_score(y_true, y_pred, *[, ...])	R^2 (coefficient of determination) regression score function.
_poisson_deviance(y_true, y_pred, *)	Mean Poisson deviance regression loss.

Toggle Menu

metrics.mean_gamma_deviance(y_true, y_pred, *)	Mean Gamma deviance regression loss.
metrics.mean_tweedie_deviance(y_true, y_pred, *)	Mean Tweedie deviance regression loss.

Multilabel ranking metrics

See the Multilabel ranking metrics section of the user guide for further details.

metrics.coverage_error(y_true, y_score, *[, ...])	Coverage error measure.
metrics.label_ranking_average_precision_score(...)	Compute ranking-based average precision.
metrics.label_ranking_loss(y_true, y_score, *)	Compute Ranking loss measure.

Clustering metrics

See the Clustering performance evaluation section of the user guide for further details.

The sklearn.metrics.cluster submodule contains evaluation metrics for cluster analysis results. There are two forms of evaluation:

- supervised, which uses a ground truth class values for each sample.
- unsupervised, which does not and measures the ‘quality’ of the model itself.

metrics.adjusted_mutual_info_score(...[, ...])	Adjusted Mutual Information between two clusterings.
metrics.adjusted_rand_score(labels_true, ...)	Rand index adjusted for chance.
metrics.calinski_harabasz_score(X, labels)	Compute the Calinski and Harabasz score.
metrics.davies_bouldin_score(X, labels)	Computes the Davies-Bouldin score.
metrics.completeness_score(labels_true, ...)	Completeness metric of a cluster labeling given a ground truth.
metrics.cluster.contingency_matrix(...[, ...])	Build a contingency matrix describing the relationship between labels.
metrics.cluster.pair_confusion_matrix(...)	Pair confusion matrix arising from two clusterings.
metrics.fowlkes_mallows_score(labels_true, ...)	Measure the similarity of two clusterings of a set of points.
metrics.homogeneity_completeness_v_measure(...)	Compute the homogeneity and completeness and V-Measure scores at once.
metrics.homogeneity_score(labels_true, ...)	Homogeneity metric of a cluster labeling given a ground truth.
metrics.mutual_info_score(labels_true, ...)	Mutual Information between two clusterings.
metrics.normalized_mutual_info_score(...[, ...])	Normalized Mutual Information between two clusterings.
metrics.rand_score(labels_true, labels_pred)	Rand index.
metrics.silhouette_score(X, labels, *[, ...])	Compute the mean Silhouette Coefficient of all samples.
metrics.silhouette_samples(X, labels, *[, ...])	Compute the Silhouette Coefficient for each sample.
metrics.v_measure_score(labels_true, ...[, beta])	V-measure cluster labeling given a ground truth.

Biclustering metrics

See the Biclustering evaluation section of the user guide for further details.

metrics.consensus_score(a, b, *[, similarity])	The similarity of two sets of biclusters.

Pairwise metrics

See the Pairwise metrics, Affinities and Kernels section of the user guide for further details.

metrics.pairwise.additive_chi2_kernel(X[, Y])	Computes the additive chi-squared kernel between observations in X and Y.
metrics.pairwise.chi2_kernel(X[, Y, gamma])	Computes the exponential chi-squared kernel X and Y.
metrics.pairwise.cosine_similarity(X[, Y, ...])	Compute cosine similarity between samples in X and Y.
metrics.pairwise.cosine_distances(X[, Y])	Compute cosine distance between samples in X and Y.
metrics.pairwise.distance_metrics()	Valid metrics for pairwise_distances.
metrics.pairwise.euclidean_distances(X[, Y, ...])	Considering the rows of X (and Y=X) as vectors, compute the distance matrix between each pair of vectors.
metrics.pairwise.haversine_distances(X[, Y])	Compute the Haversine distance between samples in X and Y.
metrics.pairwise.kernel_metrics()	Valid metrics for pairwise_kernels.
metrics.pairwise.laplacian_kernel(X[, Y, gamma])	Compute the laplacian kernel between X and Y.
metrics.pairwise.linear_kernel(X[, Y, ...])	Compute the linear kernel between X and Y.
metrics.pairwise.manhattan_distances(X[, Y, ...])	Compute the L1 distances between the vectors in X and Y.
metrics.pairwise.nan_euclidean_distances(X)	Calculate the euclidean distances in the presence of missing values.
metrics.pairwise.pairwise_kernels(X[, Y, ...])	Compute the kernel between arrays X and optional array Y.
metrics.pairwise.polynomial_kernel(X[, Y, ...])	Compute the polynomial kernel between X and Y.
metrics.pairwise.rbf_kernel(X[, Y, gamma])	Compute the rbf (gaussian) kernel between X and Y.
metrics.pairwise.sigmoid_kernel(X[, Y, ...])	Compute the sigmoid kernel between X and Y.
metrics.pairwise.paired_euclidean_distances(X, Y)	Computes the paired euclidean distances between X and Y.
metrics.pairwise.paired_manhattan_distances(X, Y)	Compute the L1 distances between the vectors in X and Y.
metrics.pairwise.paired_cosine_distances(X, Y)	Computes the paired cosine distances between X and Y.
Toggle Menu wise.paired_distances(X, Y, *[, ...])	Computes the paired distances between X and Y.

metrics.pairwise_distances(X[, Y, metric, ...])	Compute the distance matrix from a vector array X and optional Y.
metrics.pairwise_distances_argmin(X, Y, *[, ...])	Compute minimum distances between one point and a set of points.
metrics.pairwise_distances_argmin_min(X, Y, *)	Compute minimum distances between one point and a set of points.
metrics.pairwise_distances_chunked(X[, Y, ...])	Generate a distance matrix chunk by chunk with optional reduction.

Plotting

See the Visualizations section of the user guide for further details.

metrics.plot_confusion_matrix(estimator, X, ...)	Plot Confusion Matrix.
metrics.plot_det_curve(estimator, X, y, *[, ...])	Plot detection error tradeoff (DET) curve.
metrics.plot_precision_recall_curve(...[, ...])	Plot Precision Recall Curve for binary classifiers.
metrics.plot_roc_curve(estimator, X, y, *[, ...])	Plot Receiver operating characteristic (ROC) curve.

metrics.ConfusionMatrixDisplay(...[, ...])	Confusion Matrix visualization.
metrics.DetCurveDisplay(*, fpr, fnr[, ...])	DET curve visualization.
metrics.PrecisionRecallDisplay(precision, ...)	Precision Recall visualization.
metrics.RocCurveDisplay(*, fpr, tpr[, ...])	ROC Curve visualization.

sklearn.mixture: Gaussian Mixture Models

The sklearn.mixture module implements mixture modeling algorithms.

User guide: See the Gaussian mixture models section for further details.

mixture.BayesianGaussianMixture(*[, ...])	Variational Bayesian estimation of a Gaussian mixture.
mixture.GaussianMixture([n_components, ...])	Gaussian Mixture.

sklearn.model_selection: Model Selection

User guide: See the Cross-validation: evaluating estimator performance, Tuning the hyper-parameters of an estimator and Learning curve sections for further details.

Splitter Classes

model_selection.GroupKFold([n_splits])	K-fold iterator variant with non-overlapping groups.
model_selection.GroupShuffleSplit([...])	Shuffle-Group(s)-Out cross-validation iterator
model_selection.KFold([n_splits, shuffle, ...])	K-Folds cross-validator
model_selection.LeaveOneGroupOut()	Leave One Group Out cross-validator
model_selection.LeavePGroupsOut(n_groups)	Leave P Group(s) Out cross-validator
model_selection.LeaveOneOut()	Leave-One-Out cross-validator
model_selection.LeavePOut(p)	Leave-P-Out cross-validator
model_selection.PredefinedSplit(test_fold)	Predefined split cross-validator
model_selection.RepeatedKFold(*[, n_splits, ...])	Repeated K-Fold cross validator.
model_selection.RepeatedStratifiedKFold(*[, ...])	Repeated Stratified K-Fold cross validator.
model_selection.ShuffleSplit([n_splits, ...])	Random permutation cross-validator
model_selection.StratifiedKFold([n_splits, ...])	Stratified K-Folds cross-validator.
model_selection.StratifiedShuffleSplit([...])	Stratified ShuffleSplit cross-validator
model_selection.TimeSeriesSplit([n_splits, ...])	Time Series cross-validator

Splitter Functions

model_selection.check_cv([cv, y, classifier])	Input checker utility for building a cross-validator
model_selection.train_test_split(*arrays[, ...])	Split arrays or matrices into random train and test subsets

Hyper-parameter optimizers

model_selection.GridSearchCV(estimator, ...)	Exhaustive search over specified parameter values for an estimator.
model_selection.HalvingGridSearchCV(...[, ...])	Search over specified parameter values with successive halving.
model_selection.ParameterGrid(param_grid)	Grid of parameters with a discrete number of values for each.
model_selection.ParameterSampler(...[, ...])	Generator on parameters sampled from given distributions.
model_selection.RandomizedSearchCV(...[, ...])	Randomized search on hyper parameters.
model_selection.HalvingRandomSearchCV(...[, ...])	Randomized search on hyper parameters.

Toggle Menu

ation

model_selection.cross_validate(estimator, X)	Evaluate metric(s) by cross-validation and also record fit/score times.
model_selection.cross_val_predict(estimator, X)	Generate cross-validated estimates for each input data point
model_selection.cross_val_score(estimator, X)	Evaluate a score by cross-validation
model_selection.learning_curve(estimator, X, ...)	Learning curve.
model_selection.permutation_test_score(...)	Evaluate the significance of a cross-validated score with permutations
model_selection.validation_curve(estimator, ...)	Validation curve.

sklearn.multiclass: Multiclass classification

Multiclass classification strategies

This module implements multiclass learning algorithms:

- one-vs-the-rest / one-vs-all
- one-vs-one
- error correcting output codes

The estimators provided in this module are meta-estimators: they require a base estimator to be provided in their constructor. For example, it is possible to use these estimators to turn a binary classifier or a regressor into a multiclass classifier. It is also possible to use these estimators with multiclass estimators in the hope that their accuracy or runtime performance improves.

All classifiers in scikit-learn implement multiclass classification; you only need to use this module if you want to experiment with custom multiclass strategies.

The one-vs-the-rest meta-classifier also implements a `predict_proba` method, so long as such a method is implemented by the base classifier. This method returns probabilities of class membership in both the single label and multilabel case. Note that in the multilabel case, probabilities are the marginal probability that a given sample falls in the given class. As such, in the multilabel case the sum of these probabilities over all possible labels for a given sample *will not* sum to unity, as they do in the single label case.

User guide: See the Multiclass classification section for further details.

multiclass.OneVsRestClassifier(estimator, *)	One-vs-the-rest (OvR) multiclass strategy.
multiclass.OneVsOneClassifier(estimator, *)	One-vs-one multiclass strategy
multiclass.OutputCodeClassifier(estimator, *)	(Error-Correcting) Output-Code multiclass strategy

sklearn.multioutput: Multioutput regression and classification

This module implements multioutput regression and classification.

The estimators provided in this module are meta-estimators: they require a base estimator to be provided in their constructor. The meta-estimator extends single output estimators to multioutput estimators.

User guide: See the Multilabel classification, Multiclass-multioutput classification, and Multioutput regression sections for further details.

multioutput.ClassifierChain(base_estimator, *)	A multi-label model that arranges binary classifiers into a chain.
multioutput.MultiOutputRegressor(estimator, *)	Multi target regression
multioutput.MultiOutputClassifier(estimator, *)	Multi target classification
multioutput.RegressorChain(base_estimator, *)	A multi-label model that arranges regressions into a chain.

sklearn.naive_bayes: Naive Bayes

The `sklearn.naive_bayes` module implements Naive Bayes algorithms. These are supervised learning methods based on applying Bayes' theorem with strong (naive) feature independence assumptions.

User guide: See the Naive Bayes section for further details.

naive_bayes.BernoulliNB(*[, alpha, ...])	Naive Bayes classifier for multivariate Bernoulli models.
naive_bayes.CategoricalNB(*[, alpha, ...])	Naive Bayes classifier for categorical features
naive_bayes.ComplementNB(*[, alpha, ...])	The Complement Naive Bayes classifier described in Rennie et al.
naive_bayes.GaussianNB(*[, priors, ...])	Gaussian Naive Bayes (GaussianNB)
naive_bayes.MultinomialNB(*[, alpha, ...])	Naive Bayes classifier for multinomial models

Toggle Menu

neighbors: Nearest Neighbors

The `sklearn.neighbors` module implements the k-nearest neighbors algorithm.

User guide: See the Nearest Neighbors section for further details.

`neighbors.BallTree`(X[, leaf_size, metric])	BallTree for fast generalized N-point problems
`neighbors.DistanceMetric`	DistanceMetric class
`neighbors.KDTree`(X[, leaf_size, metric])	KDTree for fast generalized N-point problems
`neighbors.KernelDensity`(*[, bandwidth, ...])	Kernel Density Estimation.
`neighbors.KNeighborsClassifier`([...])	Classifier implementing the k-nearest neighbors vote.
`neighbors.KNeighborsRegressor`([n_neighbors, ...])	Regression based on k-nearest neighbors.
`neighbors.KNeighborsTransformer`(*[, mode, ...])	Transform X into a (weighted) graph of k nearest neighbors
`neighbors.LocalOutlierFactor`([n_neighbors, ...])	Unsupervised Outlier Detection using Local Outlier Factor (LOF)
`neighbors.RadiusNeighborsClassifier`([...])	Classifier implementing a vote among neighbors within a given radius
`neighbors.RadiusNeighborsRegressor`([radius, ...])	Regression based on neighbors within a fixed radius.
`neighbors.RadiusNeighborsTransformer`(*[, ...])	Transform X into a (weighted) graph of neighbors nearer than a radius
`neighbors.NearestCentroid`([metric, ...])	Nearest centroid classifier.
`neighbors.NearestNeighbors`(*[, n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.
`neighbors.NeighborhoodComponentsAnalysis`([...])	Neighborhood Components Analysis

`neighbors.kneighbors_graph`(X, n_neighbors, *)	Computes the (weighted) graph of k-Neighbors for points in X
`neighbors.radius_neighbors_graph`(X, radius, *)	Computes the (weighted) graph of Neighbors for points in X

`sklearn.neural_network`: Neural network models

The `sklearn.neural_network` module includes models based on neural networks.

User guide: See the Neural network models (supervised) and Neural network models (unsupervised) sections for further details.

`neural_network.BernoulliRBM`([n_components, ...])	Bernoulli Restricted Boltzmann Machine (RBM).
`neural_network.MLPClassifier`([...])	Multi-layer Perceptron classifier.
`neural_network.MLPRegressor`([...])	Multi-layer Perceptron regressor.

`sklearn.pipeline`: Pipeline

The `sklearn.pipeline` module implements utilities to build a composite estimator, as a chain of transforms and estimators.

User guide: See the Pipelines and composite estimators section for further details.

`pipeline.FeatureUnion`(transformer_list, *[, ...])	Concatenates results of multiple transformer objects.
`pipeline.Pipeline`(steps, *[, memory, verbose])	Pipeline of transforms with a final estimator.

`pipeline.make_pipeline`(*steps[, memory, verbose])	Construct a Pipeline from the given estimators.
`pipeline.make_union`(*transformers[, n_jobs, ...])	Construct a FeatureUnion from the given transformers.

`sklearn.preprocessing`: Preprocessing and Normalization

The `sklearn.preprocessing` module includes scaling, centering, normalization, binarization methods.

User guide: See the Preprocessing data section for further details.

`preprocessing.Binarizer`(*[, threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold.
`preprocessing.FunctionTransformer`([func, ...])	Constructs a transformer from an arbitrary callable.
`preprocessing.KBinsDiscretizer`([n_bins, ...])	Bin continuous data into intervals.
`preprocessing.KernelCenterer`()	Center a kernel matrix.
`preprocessing.LabelBinarizer`(*[, neg_label, ...])	Binarize labels in a one-vs-all fashion.
`preprocessing.LabelEncoder`()	Encode target labels with value between 0 and n_classes-1.
`preprocessing.MultiLabelBinarizer`(*[, ...])	Transform between iterable of iterables and a multilabel format.
`preprocessing.MaxAbsScaler`(*[, copy])	Scale each feature by its maximum absolute value.
`preprocessing.MinMaxScaler`([feature_range, ...])	Transform features by scaling each feature to a given range.
`preprocessing.Normalizer`([norm, copy])	Normalize samples individually to unit norm.
`preprocessing.OneHotEncoder`(*[, categories, ...])	Encode categorical features as a one-hot numeric array.
`preprocessing.OrdinalEncoder`(*[, ...])	Encode categorical features as an integer array.
`preprocessing.PolynomialFeatures`([degree, ...])	Generate polynomial and interaction features.
`preprocessing.PowerTransformer`([method, ...])	Apply a power transform featurewise to make data more Gaussian-like.
`preprocessing.QuantileTransformer`(*[, ...])	Transform features using quantiles information.

Toggle Menu

preprocessing.RobustScaler(*[, ...])	Scale features using statistics that are robust to outliers.
preprocessing.StandardScaler(*[, copy, ...])	Standardize features by removing the mean and scaling to unit variance

preprocessing.add_dummy_feature(X[, value])	Augment dataset with an additional dummy feature.
preprocessing.binarize(X, *[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix.
preprocessing.label_binarize(y, *, classes)	Binarize labels in a one-vs-all fashion.
preprocessing.maxabs_scale(X, *[, axis, copy])	Scale each feature to the [-1, 1] range without breaking the sparsity.
preprocessing.minmax_scale(X[, ...])	Transform features by scaling each feature to a given range.
preprocessing.normalize(X[, norm, axis, ...])	Scale input vectors individually to unit norm (vector length).
preprocessing.quantile_transform(X, *[, ...])	Transform features using quantiles information.
preprocessing.robust_scale(X, *[, axis, ...])	Standardize a dataset along any axis
preprocessing.scale(X, *[, axis, with_mean, ...])	Standardize a dataset along any axis.
preprocessing.power_transform(X[, method, ...])	Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.

sklearn.random_projection: Random projection

Random Projection transformers.

Random Projections are a simple and computationally efficient way to reduce the dimensionality of the data by trading a controlled amount of accuracy (as additional variance) for faster processing times and smaller model sizes.

The dimensions and distribution of Random Projections matrices are controlled so as to preserve the pairwise distances between any two samples of the dataset.

The main theoretical result behind the efficiency of random projection is the Johnson-Lindenstrauss lemma (quoting Wikipedia):

> In mathematics, the Johnson-Lindenstrauss lemma is a result concerning low-distortion embeddings of points from high-dimensional into low-dimensional Euclidean space. The lemma states that a small set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved. The map used for the embedding is at least Lipschitz, and can even be taken to be an orthogonal projection.

User guide: See the Random Projection section for further details.

random_projection.GaussianRandomProjection([...])	Reduce dimensionality through Gaussian random projection.
random_projection.SparseRandomProjection([...])	Reduce dimensionality through sparse random projection.

random_projection.johnson_lindenstrauss_min_dim(...)	Find a 'safe' number of components to randomly project to.

sklearn.semi_supervised: Semi-Supervised Learning

The sklearn.semi_supervised module implements semi-supervised learning algorithms. These algorithms utilize small amounts of labeled data and large amounts of unlabeled data for classification tasks. This module includes Label Propagation.

User guide: See the Semi-supervised learning section for further details.

semi_supervised.LabelPropagation([kernel, ...])	Label Propagation classifier
semi_supervised.LabelSpreading([kernel, ...])	LabelSpreading model for semi-supervised learning
semi_supervised.SelfTrainingClassifier(...)	Self-training classifier.

sklearn.svm: Support Vector Machines

The sklearn.svm module includes Support Vector Machine algorithms.

User guide: See the Support Vector Machines section for further details.

Estimators

svm.LinearSVC([penalty, loss, dual, tol, C, ...])	Linear Support Vector Classification.
svm.LinearSVR(*[, epsilon, tol, C, loss, ...])	Linear Support Vector Regression.
svm.NuSVC(*[, nu, kernel, degree, gamma, ...])	Nu-Support Vector Classification.
svm.NuSVR(*[, nu, C, kernel, degree, gamma, ...])	Nu Support Vector Regression.
svm.OneClassSVM(*[, kernel, degree, gamma, ...])	Unsupervised Outlier Detection.
svm.SVC(*[, C, kernel, degree, gamma, ...])	C-Support Vector Classification.
...ee, gamma, coef0, ...])	Epsilon-Support Vector Regression.

Toggle Menu

svm.l1_min_c(X, y, *[, loss, fit_intercept, ...])	Return the lowest bound for C such that for C in (l1_min_C, infinity) the model is guaranteed not to be empty.

sklearn.tree: Decision Trees

The sklearn.tree module includes decision tree-based models for classification and regression.

User guide: See the Decision Trees section for further details.

tree.DecisionTreeClassifier(*[, criterion, ...])	A decision tree classifier.
tree.DecisionTreeRegressor(*[, criterion, ...])	A decision tree regressor.
tree.ExtraTreeClassifier(*[, criterion, ...])	An extremely randomized tree classifier.
tree.ExtraTreeRegressor(*[, criterion, ...])	An extremely randomized tree regressor.

tree.export_graphviz(decision_tree[, ...])	Export a decision tree in DOT format.
tree.export_text(decision_tree, *[, ...])	Build a text report showing the rules of a decision tree.

Plotting

tree.plot_tree(decision_tree, *[, ...])	Plot a decision tree.

sklearn.utils: Utilities

The sklearn.utils module includes various utilities.

Developer guide: See the Utilities for Developers page for further details.

utils.arrayfuncs.min_pos	Find the minimum value of an array over positive values
utils.as_float_array(X, *[, copy, ...])	Converts an array-like to an array of floats.
utils.assert_all_finite(X, *[, allow_nan])	Throw a ValueError if X contains NaN or infinity.
utils.Bunch(**kwargs)	Container object exposing keys as attributes.
utils.check_X_y(X, y[, accept_sparse, ...])	Input validation for standard estimators.
utils.check_array(array[, accept_sparse, ...])	Input validation on an array, list, sparse matrix or similar.
utils.check_scalar(x, name, target_type, *)	Validate scalar parameters type and value.
utils.check_consistent_length(*arrays)	Check that all arrays have consistent first dimensions.
utils.check_random_state(seed)	Turn seed into a np.random.RandomState instance
utils.class_weight.compute_class_weight(...)	Estimate class weights for unbalanced datasets.
utils.class_weight.compute_sample_weight(...)	Estimate sample weights by class for unbalanced datasets.
utils.deprecated([extra])	Decorator to mark a function or class as deprecated.
utils.estimator_checks.check_estimator(Estimator)	Check if estimator adheres to scikit-learn conventions.
utils.estimator_checks.parametrize_with_checks(...)	Pytest specific decorator for parametrizing estimator checks.
utils.estimator_html_repr(estimator)	Build a HTML representation of an estimator.
utils.extmath.safe_sparse_dot(a, b, *[, ...])	Dot product that handle the sparse matrix case correctly.
utils.extmath.randomized_range_finder(A, *, ...)	Computes an orthonormal matrix whose range approximates the range of A.
utils.extmath.randomized_svd(M, n_components, *)	Computes a truncated randomized SVD.
utils.extmath.fast_logdet(A)	Compute log(det(A)) for A symmetric.
utils.extmath.density(w, **kwargs)	Compute density of a sparse vector.
utils.extmath.weighted_mode(a, w, *[, axis])	Returns an array of the weighted modal (most common) value in a.
utils.gen_batches(n, batch_size, *[, ...])	Generator to create slices containing batch_size elements, from 0 to n.
utils.gen_even_slices(n, n_packs, *[, n_samples])	Generator to create n_packs slices going up to n.
utils.graph.single_source_shortest_path_length(...)	Return the shortest path length from source to all reachable nodes.
utils.graph_shortest_path.graph_shortest_path	Perform a shortest-path graph search on a positive directed or undirected graph.
utils.indexable(*iterables)	Make arrays indexable for cross-validation.
utils.metaestimators.if_delegate_has_method(...)	Create a decorator for methods that are delegated to a sub-estimator
utils.multiclass.type_of_target(y)	Determine the type of data indicated by the target.
utils.multiclass.is_multilabel(y)	Check if y is in a multilabel format.
utils.multiclass.unique_labels(*ys)	Extract an ordered array of unique labels.
utils.murmurhash3_32	Compute the 32bit murmurhash3 of key at seed.
utils.resample(*arrays[, replace, ...])	Resample arrays or sparse matrices in a consistent way.
utils._safe_indexing(X, indices, *[, axis])	Return rows, items or columns of X using indices.
utils.safe_mask(X, mask)	Return a mask which is safe to use on X.

Toggle Menu

utils.safe_sqr(X, *[, copy])	Element wise squaring of array-likes and sparse matrices.
utils.shuffle(*arrays[, random_state, n_samples])	Shuffle arrays or sparse matrices in a consistent way.
utils.sparsefuncs.incr_mean_variance_axis(X, …)	Compute incremental mean and variance along an axis on a CSR or CSC matrix.
utils.sparsefuncs.inplace_column_scale(X, scale)	Inplace column scaling of a CSC/CSR matrix.
utils.sparsefuncs.inplace_row_scale(X, scale)	Inplace row scaling of a CSR or CSC matrix.
utils.sparsefuncs.inplace_swap_row(X, m, n)	Swaps two rows of a CSC/CSR matrix in-place.
utils.sparsefuncs.inplace_swap_column(X, m, n)	Swaps two columns of a CSC/CSR matrix in-place.
utils.sparsefuncs.mean_variance_axis(X, axis)	Compute mean and variance along an axis on a CSR or CSC matrix.
utils.sparsefuncs.inplace_csr_column_scale(X, …)	Inplace column scaling of a CSR matrix.
utils.sparsefuncs_fast.inplace_csr_row_normalize_l1	Inplace row normalize using the l1 norm
utils.sparsefuncs_fast.inplace_csr_row_normalize_l2	Inplace row normalize using the l2 norm
utils.random.sample_without_replacement	Sample integers without replacement.
utils.validation.check_is_fitted(estimator)	Perform is_fitted validation for estimator.
utils.validation.check_memory(memory)	Check that `memory` is joblib.Memory-like.
utils.validation.check_symmetric(array, *[, …])	Make sure that array is 2D, square and symmetric.
utils.validation.column_or_1d(y, *[, warn])	Ravel column or 1d numpy array, else raises an error.
utils.validation.has_fit_parameter(…)	Checks whether the estimator's fit method supports the given parameter.
utils.all_estimators([type_filter])	Get a list of all estimators from sklearn.

Utilities from joblib:

utils.parallel_backend(backend[, n_jobs, …])	Change the default backend used by Parallel inside a with block.
utils.register_parallel_backend(name, factory)	Register a new Parallel backend factory.

Recently deprecated

To be removed in 1.0 (renaming of 0.25)

© 2007 - 2020, scikit-learn developers (BSD License). Show this page source