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.

»

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.ClassifierMixin	Mixin class for all classifiers in scikit-learn.
base.ClusterMixin	Mixin class for all cluster 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.

Functions

base.clone(estimator[, safe])	Constructs a new estimator with the same parameters.

sklearn.cluster: Clustering

The sklearn.cluster module gathers popular unsupervised clustering algorithms.

User guide: See the Clustering section for further details.

Classes

cluster.AffinityPropagation([damping, ...])	Perform Affinity Propagation Clustering of data.
cluster.AgglomerativeClustering([...])	Agglomerative Clustering
cluster.Birch([threshold, branching_factor, ...])	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.SpectralClustering([n_clusters, ...])	Apply clustering to a projection to the normalized laplacian.

Functions

cluster.estimate_bandwidth(X[, quantile, ...])	Estimate the bandwidth to use with the mean-shift algorithm.
cluster.k_means(X, n_clusters[, init, ...])	K-means clustering algorithm.
cluster.ward_tree(X[, connectivity, ...])	Ward clustering based on a Feature matrix.
cluster.affinity_propagation(S[, ...])	Perform Affinity Propagation Clustering of data
cluster.dbscan(X[, eps, min_samples, ...])	Perform DBSCAN clustering from vector array or distance matrix.
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 to the normalized laplacian.

sklearn.cluster.bicluster: Biclustering

Spectral biclustering algorithms.

Authors : Kemal Eren License: BSD 3 clause

User guide: See the Biclustering section for further details.

Classes

SpectralBiclustering([n_clusters, method, ...])	Spectral biclustering (Kluger, 2003).
SpectralCoclustering([n_clusters, ...])	Spectral Co-Clustering algorithm (Dhillon, 2001).

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.GraphLasso([alpha, mode, tol, ...])	Sparse inverse covariance estimation with an l1-penalized estimator.
covariance.GraphLassoCV([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.ledoit_wolf(X[, assume_centered, ...])	Estimates the shrunk Ledoit-Wolf covariance matrix.
covariance.shrunk_covariance(emp_cov[, ...])	Calculates a covariance matrix shrunk on the diagonal
covariance.oas(X[, assume_centered])	Estimate covariance with the Oracle Approximating Shrinkage algorithm.
covariance.graph_lasso(emp_cov, alpha[, ...])	l1-penalized covariance estimator

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.KFold([n_splits, shuffle, ...])	K-Folds cross-validator
model_selection.GroupKFold([n_splits])	K-fold iterator variant with non-overlapping groups.
model_selection.StratifiedKFold([n_splits, ...])	Stratified 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.ShuffleSplit([n_splits, ...])	Random permutation cross-validator
model_selection.GroupShuffleSplit([...])	Shuffle-Group(s)-Out cross-validation iterator
model_selection.StratifiedShuffleSplit([...])	Stratified ShuffleSplit cross-validator
model_selection.PredefinedSplit(test_fold)	Predefined split cross-validator
model_selection.TimeSeriesSplit([n_splits])	Time Series cross-validator

Splitter Functions

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

Hyper-parameter optimizers

model_selection.GridSearchCV(estimator, ...)	Exhaustive search over specified parameter values for an estimator.
model_selection.RandomizedSearchCV(... [, ...])	Randomized search on hyper parameters.
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.fit_grid_point(X, y, ... [, ...])	Run fit on one set of parameters.

Model validation

model_selection.cross_val_score(estimator, X)	Evaluate a score by cross-validation
model_selection.cross_val_predict(estimator, X)	Generate cross-validated estimates for each input data point
model_selection.permutation_test_score(...)	Evaluate the significance of a cross-validated score with permutations
model_selection.learning_curve(estimator, X, y)	Learning curve.
model_selection.validation_curve(estimator, ...)	Validation curve.

» sklearn.datasets: Datasets

The sklearn.datasets module includes utilities to load datasets, including methods to load and fetch popular reference datasets. It also features some 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.get_data_home([data_home])	Return the path of the scikit-learn data dir.
datasets.fetch_20newsgroups([data_home, ...])	Load the filenames and data from the 20 newsgroups dataset.
datasets.fetch_20newsgroups_vectorized([...])	Load the 20 newsgroups dataset and transform it into tf-idf vectors.
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])	Load and return the diabetes dataset (regression).
datasets.load_digits([n_class, return_X_y])	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])	Load and return the iris dataset (classification).
datasets.fetch_lfw_pairs([subset, ...])	Loader for the Labeled Faces in the Wild (LFW) pairs dataset
datasets.fetch_lfw_people([data_home, ...])	Loader for the Labeled Faces in the Wild (LFW) people dataset
datasets.load_linnerud([return_X_y])	Load and return the linnerud dataset (multivariate regression).
datasets.mldata_filename(dataname)	Convert a raw name for a data set in a mldata.org filename.
datasets.fetch_mldata(dataname[, ...])	Fetch an mldata.org data set
datasets.fetch_olivetti_faces([data_home, ...])	Loader for the Olivetti faces data-set from AT&T.
datasets.fetch_california_housing([...])	Loader for the California housing dataset from StatLib.
datasets.fetch_covtype([data_home, ...])	Load the covertype dataset, downloading it if necessary.
datasets.fetch_kddcup99([subset, shuffle, ...])	Load and return the kddcup 99 dataset (classification).
datasets.fetch_rcv1([data_home, subset, ...])	Load the RCV1 multilabel dataset, downloading it if necessary.
datasets.load_mlcomp(name_or_id[, set_, ...])	Load a datasets as downloaded from http://mlcomp.org
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.fetch_species_distributions([...])	Loader for species distribution dataset from Phillips et.
datasets.load_svmlight_file(f[, n_features, ...])	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.dump_svmlight_file(X, y, f[, ...])	Dump the dataset in svmlight / libsvm file format.

Samples generator

datasets.make_blobs([n_samples, n_features, ...])	Generate isotropic Gaussian blobs for clustering.
datasets.make_classification([n_samples, ...])	Generate a random n-class classification problem.
datasets.make_circles([n_samples, shuffle, ...])	Make a large circle containing a smaller circle in 2d.
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[, random_state])	Generate a random symmetric, positive-definite matrix.

datasets.make_swiss_roll([n_samples, noise, ...])	Generate a swiss roll dataset.
datasets.make_biclusters(shape, n_clusters)	Generate an array with constant block diagonal structure for biclustering.
datasets.make_checkerboard(shape, n_clusters)	Generate an array with block checkerboard structure for biclustering.

sklearn.decomposition: Matrix Decomposition

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

»

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

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

sklearn.dummy: Dummy estimators

User guide: See the Model evaluation: 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, constant, ...])	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([loss, ...])	Gradient Boosting for classification.
ensemble.GradientBoostingRegressor([loss, ...])	Gradient Boosting for regression.
ensemble.IsolationForest([n_estimators, ...])	Isolation Forest Algorithm
ensemble.RandomForestClassifier([...])	A random forest classifier.
ensemble.RandomTreesEmbedding([...])	An ensemble of totally random trees.
ensemble.RandomForestRegressor([...])	A random forest regressor.
ensemble.VotingClassifier(estimators[, ...])	Soft Voting/Majority Rule classifier for unfitted estimators.

partial dependence

Partial dependence plots for tree ensembles.

ensemble.partial_dependence.partial_dependence(...)	Partial dependence of target_variables.
ensemble.partial_dependence.plot_partial_dependence(...)	Partial dependence plots for features.

sklearn.exceptions: Exceptions and warnings

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

exceptions.NotFittedError	Exception class to raise if estimator is used before fitting.
exceptions.ChangedBehaviorWarning	Warning class used to notify the user of any change in the behavior.
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.NonBLASDotWarning	Warning used when the dot operation does not use BLAS.
exceptions.UndefinedMetricWarning	Warning used when the metric is invalid

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([dtype, ...])	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.img_to_graph(img[, ...])	Graph of the pixel-to-pixel gradient connections
feature_extraction.image.grid_to_graph(n_x, n_y)	Graph of the pixel-to-pixel connections
feature_extraction.image.extract_patches_2d(...)	Reshape a 2D image into a collection of patches
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.
feature_selection.SelectFwe([score_func, alpha])	Filter: Select the p-values corresponding to Family-wise error rate
feature_selection.RFE(estimator[, ...])	Feature ranking with recursive feature elimination.
feature_selection.RFECV(estimator[, step, ...])	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.GaussianProcessRegressor([...])	Gaussian process regression (GPR).
gaussian_process.GaussianProcessClassifier([...])	Gaussian process classification (GPC) based on Laplace approximation.

Kernels:

gaussian_process.kernels.Kernel	Base class for all kernels.
gaussian_process.kernels.Sum(k1, k2)	Sum-kernel k1 + k2 of two kernels k1 and k2.
gaussian_process.kernels.Product(k1, k2)	Product-kernel k1 * k2 of two kernels k1 and k2.
gaussian_process.kernels.Exponentiation(...)	Exponentiate kernel by given exponent.
gaussian_process.kernels.ConstantKernel([...])	Constant kernel.
gaussian_process.kernels.WhiteKernel([...])	White kernel.
gaussian_process.kernels.RBF([length_scale, ...])	Radial-basis function kernel (aka squared-exponential kernel).
gaussian_process.kernels.Matern([...])	Matern kernel.
gaussian_process.kernels.RationalQuadratic([...])	Rational Quadratic kernel.
gaussian_process.kernels.ExpSineSquared([...])	Exp-Sine-Squared kernel.
gaussian_process.kernels.DotProduct([...])	Dot-Product kernel.
gaussian_process.kernels.PairwiseKernel([...])	Wrapper for kernels in sklearn.metrics.pairwise.
gaussian_process.kernels.CompoundKernel(kernels)	Kernel which is composed of a set of other kernels.
gaussian_process.kernels.Hyperparameter	A kernel hyperparameter's specification in form of a namedtuple.

sklearn.isotonic: Isotonic regression

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

isotonic.IsotonicRegression([y_min, y_max, ...])	Isotonic regression model.
isotonic.isotonic_regression(y[, ...])	Solve the isotonic regression model:
isotonic.check_increasing(x, y)	Determine whether y is monotonically correlated with x.

sklearn.kernel_approximation Kernel Approximation

The sklearn.kernel_approximation module implements several approximate kernel feature maps base on Fourier transforms.

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.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.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.linear_model: Generalized Linear Models

The sklearn.linear_model module implements generalized linear models. It includes Ridge regression, Bayesian Regression, Lasso and Elastic Net estimators computed with Least Angle Regression and coordinate descent. It also implements Stochastic Gradient

Descent related algorithms.

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

linear_model.ARDRegression([n_iter, tol, ...])	Bayesian ARD regression.
linear_model.BayesianRidge([n_iter, tol, ...])	Bayesian ridge regression
linear_model.ElasticNet([alpha, l1_ratio, ...])	Linear regression with combined L1 and L2 priors as regularizer.
linear_model.ElasticNetCV([l1_ratio, eps, ...])	Elastic Net model with iterative fitting along a regularization path
linear_model.HuberRegressor([epsilon, ...])	Linear regression model that is robust to outliers.
linear_model.Lars([fit_intercept, verbose, ...])	Least Angle Regression model a.k.a.
» linear_model.LarsCV([fit_intercept, ...])	Cross-validated Least Angle Regression model
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.LinearRegression([...])	Ordinary least squares Linear Regression.
linear_model.LogisticRegression([penalty, ...])	Logistic Regression (aka logit, MaxEnt) classifier.
linear_model.LogisticRegressionCV([Cs, ...])	Logistic Regression CV (aka logit, MaxEnt) classifier.
linear_model.MultiTaskLasso([alpha, ...])	Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer
linear_model.MultiTaskElasticNet([alpha, ...])	Multi-task ElasticNet model trained with L1/L2 mixed-norm as regularizer
linear_model.MultiTaskLassoCV([eps, ...])	Multi-task L1/L2 Lasso with built-in cross-validation.
linear_model.MultiTaskElasticNetCV([...])	Multi-task L1/L2 ElasticNet with built-in cross-validation.
linear_model.OrthogonalMatchingPursuit([...])	Orthogonal Matching Pursuit model (OMP)
linear_model.OrthogonalMatchingPursuitCV([...])	Cross-validated Orthogonal Matching Pursuit model (OMP)
linear_model.PassiveAggressiveClassifier([...])	Passive Aggressive Classifier
linear_model.PassiveAggressiveRegressor([C, ...])	Passive Aggressive Regressor
linear_model.Perceptron([penalty, alpha, ...])	Read more in the User Guide.
linear_model.RandomizedLasso([alpha, ...])	Randomized Lasso.
linear_model.RandomizedLogisticRegression([...])	Randomized Logistic Regression
linear_model.RANSACRegressor([...])	RANSAC (RANdom SAmple Consensus) algorithm.
linear_model.Ridge([alpha, fit_intercept, ...])	Linear least squares with l2 regularization.
linear_model.RidgeClassifier([alpha, ...])	Classifier using Ridge regression.
linear_model.RidgeClassifierCV([alphas, ...])	Ridge classifier with built-in cross-validation.
linear_model.RidgeCV([alphas, ...])	Ridge regression with built-in cross-validation.
linear_model.SGDClassifier([loss, penalty, ...])	Linear classifiers (SVM, logistic regression, a.o.) with SGD training.
linear_model.SGDRegressor([loss, penalty, ...])	Linear model fitted by minimizing a regularized empirical loss with SGD
linear_model.TheilSenRegressor([...])	Theil-Sen Estimator: robust multivariate regression model.
linear_model.lars_path(X, y[, Xy, Gram, ...])	Compute Least Angle Regression or Lasso path using LARS algorithm [1]
linear_model.lasso_path(X, y[, eps, ...])	Compute Lasso path with coordinate descent
linear_model.lasso_stability_path(X, y[, ...])	Stability path based on randomized Lasso estimates
linear_model.logistic_regression_path(X, y)	Compute a Logistic Regression model for a list of regularization parameters.
linear_model.orthogonal_mp(X, y[, ...])	Orthogonal Matching Pursuit (OMP)
linear_model.orthogonal_mp_gram(Gram, Xy[, ...])	Gram Orthogonal Matching Pursuit (OMP)

sklearn.manifold: Manifold Learning

The sklearn.manifold module implements data embedding techniques.

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

manifold.LocallyLinearEmbedding([...])	Locally Linear Embedding
manifold.Isomap([n_neighbors, n_components, ...])	Isomap 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.spectral_embedding(adjacency[, ...])	Project the sample on the first eigenvectors of the graph Laplacian.

sklearn.metrics: Metrics

See the Model evaluation: 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.make_scorer(score_func[, ...])	Make a scorer from a performance metric or loss function.
metrics.get_scorer(scoring)	

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[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
metrics.average_precision_score(y_true, y_score)	Compute average precision (AP) from prediction scores
metrics.brier_score_loss(y_true, y_prob[, ...])	Compute the Brier score.
metrics.classification_report(y_true, y_pred)	Build a text report showing the main classification metrics
metrics.cohen_kappa_score(y1, y2[, labels, ...])	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.f1_score(y_true, y_pred[, labels, ...])	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_similarity_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) for binary classes
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 Curve (AUC) from prediction scores
metrics.roc_curve(y_true, y_score[, ...])	Compute Receiver operating characteristic (ROC)
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, y_pred)	Explained variance regression score function
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.median_absolute_error(y_true, y_pred)	Median absolute error regression loss
metrics.r2_score(y_true, y_pred[, ...])	R^2 (coefficient of determination) regression score function.

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_harabaz_score(X, labels)	Compute the Calinski and Harabaz score.
metrics.completeness_score(labels_true, ...)	Completeness metric of a cluster labeling given a ground truth.
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.silhouette_score(X, labels[, ...])	Compute the mean Silhouette Coefficient of all samples.
metrics.silhouette_samples(X, labels[, metric])	Compute the Silhouette Coefficient for each sample.
metrics.v_measure_score(labels_true, labels_pred)	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.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.kernel_metrics()	Valid metrics for pairwise_kernels
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.pairwise_distances(X[, Y, ...])	Compute the distance matrix from a vector array X and optional Y.
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.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.laplacian_kernel(X[, Y, gamma])	Compute the laplacian kernel 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.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
metrics.pairwise.paired_distances(X, Y[, metric])	Computes the paired distances between X and Y.

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

sklearn.multiclass: Multiclass and multilabel classification

Multiclass and multilabel 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 and multilabel algorithms section for further details.

multiclass.OneVsRestClassifier(estimator[, ...])	One-vs-the-rest (OvR) multiclass/multilabel 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 Multiclass and multilabel algorithms section for further details.

multioutput.MultiOutputRegressor(estimator)	Multi target regression
multioutput.MultiOutputClassifier(estimator)	Multi target classification

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.GaussianNB([priors])	Gaussian Naive Bayes (GaussianNB)
naive_bayes.MultinomialNB([alpha, ...])	Naive Bayes classifier for multinomial models
naive_bayes.BernoulliNB([alpha, binarize, ...])	Naive Bayes classifier for multivariate Bernoulli models.

sklearn.neighbors: Nearest Neighbors

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

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

neighbors.NearestNeighbors([n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.
neighbors.KNeighborsClassifier([...])	Classifier implementing the k-nearest neighbors vote.
neighbors.RadiusNeighborsClassifier([...])	Classifier implementing a vote among neighbors within a given radius
neighbors.KNeighborsRegressor([n_neighbors, ...])	Regression based on k-nearest neighbors.
neighbors.RadiusNeighborsRegressor([radius, ...])	Regression based on neighbors within a fixed radius.
neighbors.NearestCentroid([metric, ...])	Nearest centroid classifier.
neighbors.BallTree	BallTree for fast generalized N-point problems
neighbors.KDTree	KDTree for fast generalized N-point problems
neighbors.LSHForest([n_estimators, radius, ...])	Performs approximate nearest neighbor search using LSH forest.
neighbors.DistanceMetric	DistanceMetric class
neighbors.KernelDensity([bandwidth, ...])	Kernel Density Estimation
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.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 sigmoid.
calibration.calibration_curve(y_true, y_prob)	Compute true and predicted probabilities for a calibration curve.

sklearn.cross_decomposition: Cross decomposition

»

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

cross_decomposition.PLSRegression([...])	PLS regression
cross_decomposition.PLSCanonical([...])	PLSCanonical implements the 2 blocks canonical PLS of the original Wold algorithm [Tenenhaus 1998] p.204, referred as PLS-C2A in [Wegelin 2000].
cross_decomposition.CCA([n_components, ...])	CCA Canonical Correlation Analysis.
cross_decomposition.PLSSVD([n_components, ...])	Partial Least Square SVD

sklearn.pipeline: Pipeline

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

pipeline.Pipeline(steps)	Pipeline of transforms with a final estimator.
pipeline.FeatureUnion(transformer_list[, ...])	Concatenates results of multiple transformer objects.
pipeline.make_pipeline(*steps)	Construct a Pipeline from the given estimators.
pipeline.make_union(*transformers)	Construct a FeatureUnion from the given transformers.

sklearn.preprocessing: Preprocessing and Normalization

The sklearn.preprocessing module includes scaling, centering, normalization, binarization and imputation 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.Imputer([missing_values, ...])	Imputation transformer for completing missing values.
preprocessing.KernelCenterer	Center a kernel matrix
preprocessing.LabelBinarizer([neg_label, ...])	Binarize labels in a one-vs-all fashion
preprocessing.LabelEncoder	Encode labels with value between 0 and n_classes-1.
preprocessing.MultiLabelBinarizer([classes, ...])	Transform between iterable of iterables and a multilabel format
preprocessing.MaxAbsScaler([copy])	Scale each feature by its maximum absolute value.
preprocessing.MinMaxScaler([feature_range, copy])	Transforms features by scaling each feature to a given range.
preprocessing.Normalizer([norm, copy])	Normalize samples individually to unit norm.
preprocessing.OneHotEncoder([n_values, ...])	Encode categorical integer features using a one-hot aka one-of-K scheme.
preprocessing.PolynomialFeatures([degree, ...])	Generate polynomial and interaction features.
preprocessing.RobustScaler([with_centering, ...])	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[, ...])	Transforms features by scaling each feature to a given range.
preprocessing.normalize(X[, norm, axis, ...])	Scale input vectors individually to unit norm (vector length).
preprocessing.robust_scale(X[, axis, ...])	Standardize a dataset along any axis
preprocessing.scale(X[, axis, with_mean, ...])	Standardize a dataset along any axis

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 utilized 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 section for further details.

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

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

Low-level methods

svm.libsvm.fit	Train the model using libsvm (low-level method)
svm.libsvm.decision_function	Predict margin (libsvm name for this is predict_values)
svm.libsvm.predict	Predict target values of X given a model (low-level method)
svm.libsvm.predict_proba	Predict probabilities
svm.libsvm.cross_validation	Binding of the cross-validation routine (low-level routine)

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	Export a decision tree in DOT format.

sklearn.utils: Utilities

The sklearn.utils module includes various utilities.

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

utils.check_random_state(seed)	Turn seed into a np.random.RandomState instance
utils.estimator_checks.check_estimator(Estimator)	Check if estimator adheres to scikit-learn conventions.
utils.resample(*arrays, **options)	Resample arrays or sparse matrices in a consistent way
utils.shuffle(*arrays, **options)	Shuffle arrays or sparse matrices in a consistent way

» Recently deprecated

To be removed in 0.19

lda.LDA([solver, shrinkage, priors, ...])	Alias for sklearn.discriminant_analysis.LinearDiscriminantAnalysis.
qda.QDA([priors, reg_param, ...])	Alias for sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.
datasets.load_lfw_pairs(*args, **kwargs)	DEPRECATED: Function 'load_lfw_pairs' has been deprecated in 0.17 and will be removed in 0.19.Use fetch_lfw_pairs(download_if_missing=False) instead.
datasets.load_lfw_people(*args, **kwargs)	DEPRECATED: Function 'load_lfw_people' has been deprecated in 0.17 and will be removed in 0.19.Use fetch_lfw_people(download_if_missing=False) instead.

To be removed in 0.20

grid_search.ParameterGrid(param_grid)	Grid of parameters with a discrete number of values for each.
grid_search.ParameterSampler(...[, random_state])	Generator on parameters sampled from given distributions.
grid_search.GridSearchCV(estimator, param_grid)	Exhaustive search over specified parameter values for an estimator.
grid_search.RandomizedSearchCV(estimator, ...)	Randomized search on hyper parameters.
cross_validation.LeaveOneOut(n)	Leave-One-Out cross validation iterator.
cross_validation.LeavePOut(n, p)	Leave-P-Out cross validation iterator
cross_validation.KFold(n[, n_folds, ...])	K-Folds cross validation iterator.
cross_validation.LabelKFold(labels[, n_folds])	K-fold iterator variant with non-overlapping labels.
cross_validation.LeaveOneLabelOut(labels)	Leave-One-Label_Out cross-validation iterator
cross_validation.LeavePLabelOut(labels, p)	Leave-P-Label_Out cross-validation iterator
cross_validation.LabelShuffleSplit(labels[, ...])	Shuffle-Labels-Out cross-validation iterator
cross_validation.StratifiedKFold(y[, ...])	Stratified K-Folds cross validation iterator
cross_validation.ShuffleSplit(n[, n_iter, ...])	Random permutation cross-validation iterator.
cross_validation.StratifiedShuffleSplit(y[, ...])	Stratified ShuffleSplit cross validation iterator
cross_validation.PredefinedSplit(test_fold)	Predefined split cross validation iterator
decomposition.RandomizedPCA(*args, **kwargs)	Principal component analysis (PCA) using randomized SVD
gaussian_process.GaussianProcess(*args, **kwargs)	The legacy Gaussian Process model class.
mixture.GMM(*args, **kwargs)	Legacy Gaussian Mixture Model
mixture.DPGMM(*args, **kwargs)	Dirichlet Process Gaussian Mixture Models
mixture.VBGMM(*args, **kwargs)	Variational Inference for the Gaussian Mixture Model
grid_search.fit_grid_point(X, y, estimator, ...)	Run fit on one set of parameters.
learning_curve.learning_curve(estimator, X, y)	Learning curve.
learning_curve.validation_curve(estimator, ...)	Validation curve.
cross_validation.cross_val_predict(estimator, X)	Generate cross-validated estimates for each input data point
cross_validation.cross_val_score(estimator, X)	Evaluate a score by cross-validation
cross_validation.check_cv(cv[, X, y, classifier])	Input checker utility for building a CV in a user friendly way.
cross_validation.permutation_test_score(...)	Evaluate the significance of a cross-validated score with permutations
cross_validation.train_test_split(*arrays, ...)	Split arrays or matrices into random train and test subsets