# **Scikit-learn**

# **Getting Started**[**¶**](https://scikit-learn.org/stable/getting_started.html#getting-started)

<https://scikit-learn.org/stable/getting_started.html>

It assumes a very basic working knowledge of machine learning practices (model fitting, predicting, cross-validation, etc.).

Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities

## **Fitting and predicting: estimator basics**

Scikit-learn provides dozens of built-in machine learning algorithms and models, called [estimators](https://scikit-learn.org/stable/glossary.html#term-estimators). Each estimator can be fitted to some data using its [fit](https://scikit-learn.org/stable/glossary.html#term-fit) method.

Here is a simple example where we fit a **[RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html" \l "sklearn.ensemble.RandomForestClassifier" \o "sklearn.ensemble.RandomForestClassifier)** to some very basic data:

**>>> from** **sklearn.ensemble** **import** RandomForestClassifier

**>>>** clf = RandomForestClassifier(random\_state=0)

**>>>** X = [[ 1, 2, 3], *# 2 samples, 3 features*

**...**  [11, 12, 13]]

**>>>** y = [0, 1] *# classes of each sample*

**>>>** clf.fit(X, y)

RandomForestClassifier(random\_state=0)

The [fit](https://scikit-learn.org/stable/glossary.html#term-fit) method generally accepts 2 inputs:

* The samples matrix (or design matrix) [X](https://scikit-learn.org/stable/glossary.html#term-X). The size of X is typically (n\_samples, n\_features), which means that samples are represented as rows and features are represented as columns.
* The target values [y](https://scikit-learn.org/stable/glossary.html#term-y) which are real numbers for regression tasks, or integers for classification (or any other discrete set of values). For unsupervised learning tasks, y does not need to be specified. y is usually a 1d array where the i th entry corresponds to the target of the i th sample (row) of X.

Both X and y are usually expected to be numpy arrays or equivalent [array-like](https://scikit-learn.org/stable/glossary.html#term-array-like) data types, though some estimators work with other formats such as sparse matrices.

Once the estimator is fitted, it can be used for predicting target values of new data. You don’t need to re-train the estimator:

**>>>** clf.predict(X) *# predict classes of the training data*

array([0, 1])

**>>>** clf.predict([[4, 5, 6], [14, 15, 16]]) *# predict classes of new data*

array([0, 1])

## **Transformers and pre-processors**

Machine learning workflows are often composed of different parts. A typical pipeline consists of a pre-processing step that transforms or imputes the data, and a final predictor that predicts target values.

In scikit-learn, pre-processors and transformers follow the same API as the estimator objects (they actually all inherit from the same BaseEstimator class). The transformer objects don’t have a [predict](https://scikit-learn.org/stable/glossary.html#term-predict) method but rather a [transform](https://scikit-learn.org/stable/glossary.html#term-transform) method that outputs a newly transformed sample matrix X:

**>>> from** **sklearn.preprocessing** **import** StandardScaler

**>>>** X = [[0, 15],

**...**  [1, -10]]

**>>>** *# scale data according to computed scaling values*

**>>>** StandardScaler().fit(X).transform(X)

array([[-1., 1.],

[ 1., -1.]])

Sometimes, you want to apply different transformations to different features: the [ColumnTransformer](https://scikit-learn.org/stable/modules/compose.html" \l "column-transformer) is designed for these use-cases.

## **Pipelines: chaining pre-processors and estimators**

Transformers and estimators (predictors) can be combined together into a single unifying object: a [**Pipeline**](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline). The pipeline offers the same API as a regular estimator: it can be fitted and used for prediction with fit and predict. As we will see later, using a pipeline will also prevent you from data leakage, i.e. disclosing some testing data in your training data.

In the following example, we [load the Iris dataset](https://scikit-learn.org/stable/datasets.html#datasets), split it into train and test sets, and compute the accuracy score of a pipeline on the test data:

**>>> from** **sklearn.preprocessing** **import** StandardScaler

**>>> from** **sklearn.linear\_model** **import** LogisticRegression

**>>> from** **sklearn.pipeline** **import** make\_pipeline

**>>> from** **sklearn.datasets** **import** load\_iris

**>>> from** **sklearn.model\_selection** **import** train\_test\_split

**>>> from** **sklearn.metrics** **import** accuracy\_score

**...**

**>>>** *# create a pipeline object*

**>>>** pipe = make\_pipeline(

**...**  StandardScaler(),

**...**  LogisticRegression()

**...** )

**...**

**>>>** *# load the iris dataset and split it into train and test sets*

**>>>** X, y = load\_iris(return\_X\_y=**True**)

**>>>** X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

**...**

**>>>** *# fit the whole pipeline*

**>>>** pipe.fit(X\_train, y\_train)

Pipeline(steps=[('standardscaler', StandardScaler()),

('logisticregression', LogisticRegression())])

**>>>** *# we can now use it like any other estimator*

**>>>** accuracy\_score(pipe.predict(X\_test), y\_test)

0.97...

## **Model evaluation**

Fitting a model to some data does not entail that it will predict well on unseen data. This needs to be directly evaluated. We have just seen the **[train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \l "sklearn.model_selection.train_test_split" \o "sklearn.model_selection.train_test_split)** helper that splits a dataset into train and test sets, but scikit-learn provides many other tools for model evaluation, in particular for [cross-validation](https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation).

We here briefly show how to perform a 5-fold cross-validation procedure, using the **[cross\_validate](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html" \l "sklearn.model_selection.cross_validate" \o "sklearn.model_selection.cross_validate)** helper. Note that it is also possible to manually iterate over the folds, use different data splitting strategies, and use custom scoring functions. Please refer to our [User Guide](https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation) for more details:

**>>> from** **sklearn.datasets** **import** make\_regression

**>>> from** **sklearn.linear\_model** **import** LinearRegression

**>>> from** **sklearn.model\_selection** **import** cross\_validate

**...**

**>>>** X, y = make\_regression(n\_samples=1000, random\_state=0)

**>>>** lr = LinearRegression()

**...**

**>>>** result = cross\_validate(lr, X, y) *# defaults to 5-fold CV*

**>>>** result['test\_score'] *# r\_squared score is high because dataset is easy*

array([1., 1., 1., 1., 1.])

## **Automatic parameter searches**

All estimators have parameters (often called hyper-parameters in the literature) that can be tuned. The generalization power of an estimator often critically depends on a few parameters. For example a **[RandomForestRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html" \l "sklearn.ensemble.RandomForestRegressor" \o "sklearn.ensemble.RandomForestRegressor)** has a n\_estimators parameter that determines the number of trees in the forest, and a max\_depth parameter that determines the maximum depth of each tree. Quite often, it is not clear what the exact values of these parameters should be since they depend on the data at hand.

Scikit-learn provides tools to automatically find the best parameter combinations (via cross-validation). In the following example, we randomly search over the parameter space of a random forest with a **[RandomizedSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html" \l "sklearn.model_selection.RandomizedSearchCV" \o "sklearn.model_selection.RandomizedSearchCV)** object. When the search is over, the **[RandomizedSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html" \l "sklearn.model_selection.RandomizedSearchCV" \o "sklearn.model_selection.RandomizedSearchCV)** behaves as a **[RandomForestRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html" \l "sklearn.ensemble.RandomForestRegressor" \o "sklearn.ensemble.RandomForestRegressor)** that has been fitted with the best set of parameters. Read more in the [User Guide](https://scikit-learn.org/stable/modules/grid_search.html#grid-search):

**>>> from** **sklearn.datasets** **import** fetch\_california\_housing

**>>> from** **sklearn.ensemble** **import** RandomForestRegressor

**>>> from** **sklearn.model\_selection** **import** RandomizedSearchCV

**>>> from** **sklearn.model\_selection** **import** train\_test\_split

**>>> from** **scipy.stats** **import** randint

**...**

**>>>** X, y = fetch\_california\_housing(return\_X\_y=**True**)

**>>>** X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

**...**

**>>>** *# define the parameter space that will be searched over*

**>>>** param\_distributions = {'n\_estimators': randint(1, 5),

**...**  'max\_depth': randint(5, 10)}

**...**

**>>>** *# now create a searchCV object and fit it to the data*

**>>>** search = RandomizedSearchCV(estimator=RandomForestRegressor(random\_state=0),

**...**  n\_iter=5,

**...**  param\_distributions=param\_distributions,

**...**  random\_state=0)

**>>>** search.fit(X\_train, y\_train)

RandomizedSearchCV(estimator=RandomForestRegressor(random\_state=0), n\_iter=5,

param\_distributions={'max\_depth': ...,

'n\_estimators': ...},

random\_state=0)

**>>>** search.best\_params\_

{'max\_depth': 9, 'n\_estimators': 4}

**>>>** *# the search object now acts like a normal random forest estimator*

**>>>** *# with max\_depth=9 and n\_estimators=4*

**>>>** search.score(X\_test, y\_test)

0.73...

**Note**

In practice, you almost always want to [search over a pipeline](https://scikit-learn.org/stable/modules/grid_search.html#composite-grid-search), instead of a single estimator. One of the main reasons is that if you apply a pre-processing step to the whole dataset without using a pipeline, and then perform any kind of cross-validation, you would be breaking the fundamental assumption of independence between training and testing data. Indeed, since you pre-processed the data using the whole dataset, some information about the test sets are available to the train sets. This will lead to over-estimating the generalization power of the estimator (you can read more in this [Kaggle post](https://www.kaggle.com/alexisbcook/data-leakage)).

Using a pipeline for cross-validation and searching will largely keep you from this common pitfall.

<https://scikit-learn.org/stable/tutorial/index.html>

# **An introduction to machine learning with scikit-learn**[**¶**](https://scikit-learn.org/stable/tutorial/basic/tutorial.html#an-introduction-to-machine-learning-with-scikit-learn)

<https://scikit-learn.org/stable/tutorial/basic/tutorial.html>

# **A tutorial on statistical-learning for scientific data processing**[**¶**](https://scikit-learn.org/stable/tutorial/statistical_inference/index.html#a-tutorial-on-statistical-learning-for-scientific-data-processing)

<https://scikit-learn.org/stable/tutorial/statistical_inference/index.html>

# **Working With Text Data**[**¶**](https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html#working-with-text-data)

<https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html>

# **Choosing the right estimator**[**¶**](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html#choosing-the-right-estimator)

<https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html>

# **User Guide**[**¶**](https://scikit-learn.org/stable/user_guide.html#user-guide)

<https://scikit-learn.org/stable/user_guide.html>

* [**1. Supervised learning**](https://scikit-learn.org/stable/supervised_learning.html)
  + [1.1. Linear Models](https://scikit-learn.org/stable/modules/linear_model.html)
  + [1.2. Linear and Quadratic Discriminant Analysis](https://scikit-learn.org/stable/modules/lda_qda.html)
  + [1.3. Kernel ridge regression](https://scikit-learn.org/stable/modules/kernel_ridge.html)
  + [1.4. Support Vector Machines](https://scikit-learn.org/stable/modules/svm.html)
  + [1.5. Stochastic Gradient Descent](https://scikit-learn.org/stable/modules/sgd.html)
  + [1.6. Nearest Neighbors](https://scikit-learn.org/stable/modules/neighbors.html)
  + [1.7. Gaussian Processes](https://scikit-learn.org/stable/modules/gaussian_process.html)
  + [1.8. Cross decomposition](https://scikit-learn.org/stable/modules/cross_decomposition.html)
  + [1.9. Naive Bayes](https://scikit-learn.org/stable/modules/naive_bayes.html)
  + [1.10. Decision Trees](https://scikit-learn.org/stable/modules/tree.html)
  + [1.11. Ensembles: Gradient boosting, random forests, bagging, voting, stacking](https://scikit-learn.org/stable/modules/ensemble.html)
  + [1.12. Multiclass and multioutput algorithms](https://scikit-learn.org/stable/modules/multiclass.html)
  + [1.13. Feature selection](https://scikit-learn.org/stable/modules/feature_selection.html)
  + [1.14. Semi-supervised learning](https://scikit-learn.org/stable/modules/semi_supervised.html)
  + [1.15. Isotonic regression](https://scikit-learn.org/stable/modules/isotonic.html)
  + [1.16. Probability calibration](https://scikit-learn.org/stable/modules/calibration.html)
  + [1.17. Neural network models (supervised)](https://scikit-learn.org/stable/modules/neural_networks_supervised.html)
* [**2. Unsupervised learning**](https://scikit-learn.org/stable/unsupervised_learning.html)
  + [2.1. Gaussian mixture models](https://scikit-learn.org/stable/modules/mixture.html)
  + [2.2. Manifold learning](https://scikit-learn.org/stable/modules/manifold.html)
  + [2.3. Clustering](https://scikit-learn.org/stable/modules/clustering.html)
  + [2.4. Biclustering](https://scikit-learn.org/stable/modules/biclustering.html)
  + [2.5. Decomposing signals in components (matrix factorization problems)](https://scikit-learn.org/stable/modules/decomposition.html)
  + [2.6. Covariance estimation](https://scikit-learn.org/stable/modules/covariance.html)
  + [2.7. Novelty and Outlier Detection](https://scikit-learn.org/stable/modules/outlier_detection.html)
  + [2.8. Density Estimation](https://scikit-learn.org/stable/modules/density.html)
  + [2.9. Neural network models (unsupervised)](https://scikit-learn.org/stable/modules/neural_networks_unsupervised.html)
* [**3. Model selection and evaluation**](https://scikit-learn.org/stable/model_selection.html)
  + [3.1. Cross-validation: evaluating estimator performance](https://scikit-learn.org/stable/modules/cross_validation.html)
  + [3.2. Tuning the hyper-parameters of an estimator](https://scikit-learn.org/stable/modules/grid_search.html)
  + [3.3. Metrics and scoring: quantifying the quality of predictions](https://scikit-learn.org/stable/modules/model_evaluation.html)
  + [3.4. Validation curves: plotting scores to evaluate models](https://scikit-learn.org/stable/modules/learning_curve.html)
* [**4. Inspection**](https://scikit-learn.org/stable/inspection.html)
  + [4.1. Partial Dependence and Individual Conditional Expectation plots](https://scikit-learn.org/stable/modules/partial_dependence.html)
  + [4.2. Permutation feature importance](https://scikit-learn.org/stable/modules/permutation_importance.html)
* [**5. Visualizations**](https://scikit-learn.org/stable/visualizations.html)
  + [5.1. Available Plotting Utilities](https://scikit-learn.org/stable/visualizations.html#available-plotting-utilities)
* [**6. Dataset transformations**](https://scikit-learn.org/stable/data_transforms.html)
  + [6.1. Pipelines and composite estimators](https://scikit-learn.org/stable/modules/compose.html)
  + [6.2. Feature extraction](https://scikit-learn.org/stable/modules/feature_extraction.html)
  + [6.3. Preprocessing data](https://scikit-learn.org/stable/modules/preprocessing.html)
  + [6.4. Imputation of missing values](https://scikit-learn.org/stable/modules/impute.html)
  + [6.5. Unsupervised dimensionality reduction](https://scikit-learn.org/stable/modules/unsupervised_reduction.html)
  + [6.6. Random Projection](https://scikit-learn.org/stable/modules/random_projection.html)
  + [6.7. Kernel Approximation](https://scikit-learn.org/stable/modules/kernel_approximation.html)
  + [6.8. Pairwise metrics, Affinities and Kernels](https://scikit-learn.org/stable/modules/metrics.html)
  + [6.9. Transforming the prediction target (**y**)](https://scikit-learn.org/stable/modules/preprocessing_targets.html)
* [**7. Dataset loading utilities**](https://scikit-learn.org/stable/datasets.html)
  + [7.1. Toy datasets](https://scikit-learn.org/stable/datasets/toy_dataset.html)
  + [7.2. Real world datasets](https://scikit-learn.org/stable/datasets/real_world.html)
  + [7.3. Generated datasets](https://scikit-learn.org/stable/datasets/sample_generators.html)
  + [7.4. Loading other datasets](https://scikit-learn.org/stable/datasets/loading_other_datasets.html)
* [**8. Computing with scikit-learn**](https://scikit-learn.org/stable/computing.html)
  + [8.1. Strategies to scale computationally: bigger data](https://scikit-learn.org/stable/computing/scaling_strategies.html)
  + [8.2. Computational Performance](https://scikit-learn.org/stable/computing/computational_performance.html)
  + [8.3. Parallelism, resource management, and configuration](https://scikit-learn.org/stable/computing/parallelism.html)
* [**9. Model persistence**](https://scikit-learn.org/stable/model_persistence.html)
  + [9.1. Python specific serialization](https://scikit-learn.org/stable/model_persistence.html#python-specific-serialization)
  + [9.2. Interoperable formats](https://scikit-learn.org/stable/model_persistence.html#interoperable-formats)
* [**10. Common pitfalls and recommended practices**](https://scikit-learn.org/stable/common_pitfalls.html)
  + [10.1. Inconsistent preprocessing](https://scikit-learn.org/stable/common_pitfalls.html#inconsistent-preprocessing)
  + [10.2. Data leakage](https://scikit-learn.org/stable/common_pitfalls.html#data-leakage)
  + [10.3. Controlling randomness](https://scikit-learn.org/stable/common_pitfalls.html#controlling-randomness)
* [**11. Dispatching**](https://scikit-learn.org/stable/dispatching.html)
  + [11.1. Array API support (experimental)](https://scikit-learn.org/stable/modules/array_api.html)

## **Under Development**

* [**1. Metadata Routing**](https://scikit-learn.org/stable/metadata_routing.html)