#sklearn.model_selection.train_test_split

#sklearn.model_selection.train_test_split(arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)


scikit
learn
Install
User Guide
API
Examples
scikit-learn
Machine Learning in Python
Getting Started
Release Highlights for 1.0
GitHub


scikit
learn

Prev Up Next

scikit-learn 1.0.2
Other versions

Please cite us if you use the software.

sklearn.model_selection.train_test_split
Examples using
sklearn.model_selection.train_test

sklearn.model_selection.train_test_split

sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None) [source]

Split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation and `next(ShuffleSplit().split(X, y))` and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

Read more in the User Guide.

Parameters:

arrays : sequence of indexables with same length / shape[0]
Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : float or int, default=None
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : float or int, default=None
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : int, RandomState instance or None, default=None
Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : bool, default=True
Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

stratify : array-like, default=None
If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User Guide.

Returns:

splitting : list, length=2 * len(arrays)
List containing train-test split of inputs.

New in version 0.16: If the input is sparse, the output will be a `scipy.sparse.csr_matrix`. Else, output type is the same as the input type.

Toggle Menu

scikit-learn
Prev Up Next
scikit-learn 1.0.2
Other versions
Please cite us if you use the software.
sklearn.model_selection.train_test_split
Examples using
sklearn.model_selection.train_test
Toggle Menu

Examples

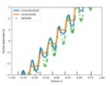
```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]
```

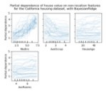
```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

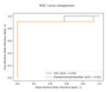
```
>>> train_test_split(y, shuffle=False)
```

```
>>> train_test_split(y, shuffle=False)
[[0, 1, 2], [3, 4]]
```

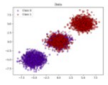
Examples using `sklearn.model_selection.train_test_split`


Release Highlights for scikit-learn 0.23

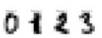

Release Highlights for scikit-learn 0.24


Release Highlights for scikit-learn 0.22


Comparison of Calibration of Classifiers

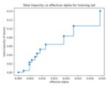

Probability calibration of classifiers

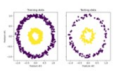

Probability Calibration curves

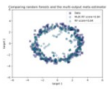

Recognizing hand-written digits

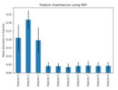

Classifier comparison

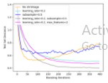

Principal Component Regression vs Partial Least Squares Regression


Post pruning decision trees with cost complexity pruning









Parameters:

arrays : *sequence of indexables with same length / shape[0]
Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

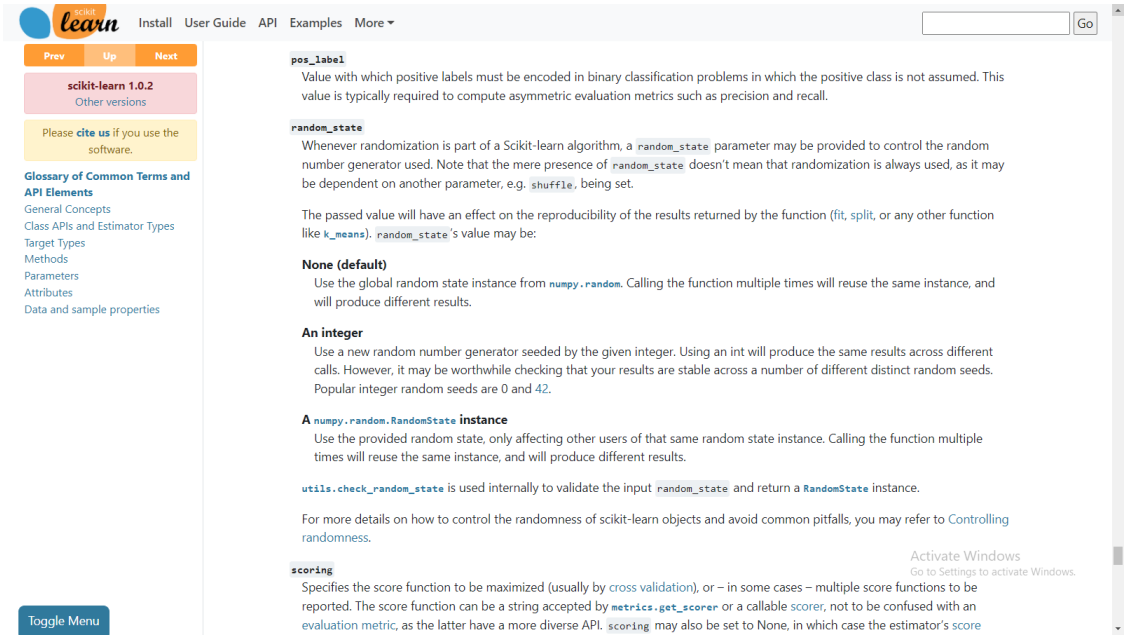
test_size : *float or int, default=None*
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : *float or int, default=None*
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : *int, RandomState instance or None, default=None*
Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : *bool, default=True*
Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

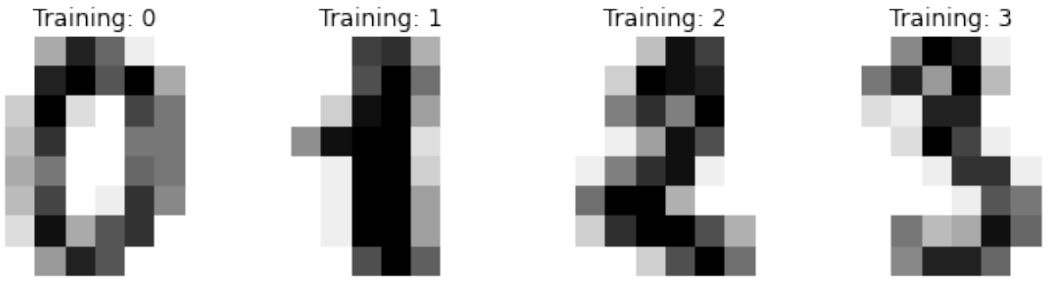
stratify : *array-like, default=None*
If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User Guide.




```
# Generalized Linear Models, and Poisson loss for gradient boosting

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import PoissonRegressor
from sklearn.ensemble import HistGradientBoostingRegressor

n_samples, n_features = 1000, 20
rng = np.random.RandomState(0)
X = rng.randn(n_samples, n_features)
# positive integer target correlated with X[:, 5] with many zeros:
y = rng.poisson(lam=np.exp(X[:, 5]) / 2)
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=rng)
glm = PoissonRegressor()
gbdt = HistGradientBoostingRegressor(loss="poisson",
learning_rate=0.01)
glm.fit(X_train, y_train)
gbdt.fit(X_train, y_train)
print('random_state=rng ', rng)
print(glm.score(X_test, y_test))
print(gbdt.score(X_test, y_test))

random_state=rng  RandomState(MT19937)
0.35776189065725783
0.42425183539869415

# Probability Calibration of Classifiers
```

```
# Author: Mathieu Blondel <mathieu@mblondel.org>
#         Alexandre Gramfort <alexandre.gramfort@telecom-paristech.fr>
#         Balazs Kegl <balazs.kegl@gmail.com>
#         Jan Hendrik Metzen <jhm@informatik.uni-bremen.de>
# License: BSD Style.

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

from sklearn.datasets import make_blobs
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import brier_score_loss
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import train_test_split


n_samples = 50000
n_bins = 3  # use 3 bins for calibration_curve as we have 3 clusters
here

# Generate 3 blobs with 2 classes where the second blob contains
# half positive samples and half negative samples. Probability in this
# blob is therefore 0.5.
centers = [(-5, -5), (0, 0), (5, 5)]
X, y = make_blobs(n_samples=n_samples, centers=centers, shuffle=False,
random_state=42)

y[: n_samples // 2] = 0
y[n_samples // 2 :] = 1
sample_weight = np.random.RandomState(42).rand(y.shape[0])

# split train, test for calibration
X_train, X_test, y_train, y_test, sw_train, sw_test =
train_test_split(
    X, y, sample_weight, test_size=0.9, random_state=42)

# Recognizing hand-written digits

# Author: Gael Varoquaux <gael dot varoquaux at normalesup dot org>
# License: BSD 3 clause

# Standard scientific Python imports
import matplotlib.pyplot as plt

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split
```

```
digits = datasets.load_digits()

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)

# flatten the images
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a classifier: a support vector classifier
clf = svm.SVC(gamma=0.001)

# Split data into 50% train and 50% test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=1/2, shuffle=False
)

# Learn the digits on the train subset
clf.fit(X_train, y_train)

# Predict the value of the digit on the test subset
predicted = clf.predict(X_test)
```

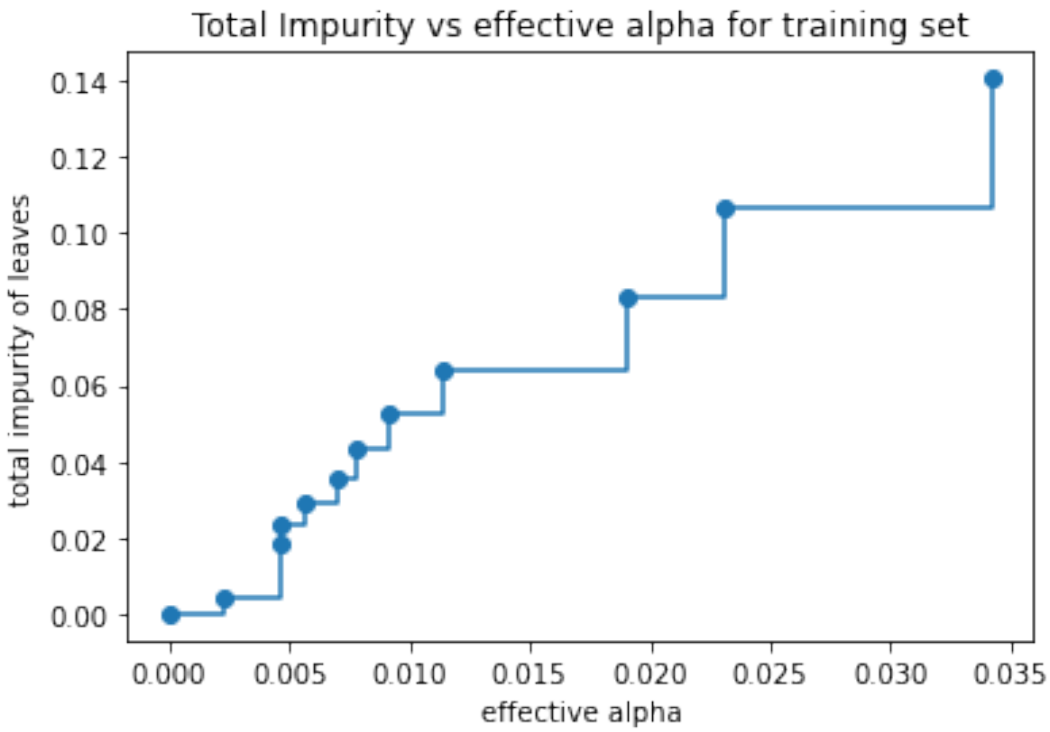



```
# Comparing random forests and the multi-output meta estimator

# Author: Tim Head <betatim@gmail.com>
#
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputRegressor


# Create a random dataset
```

```
rng = np.random.RandomState(1)
X = np.sort(200 * rng.rand(600, 1) - 100, axis=0)
y = np.array([np.pi * np.sin(X).ravel(), np.pi * np.cos(X).ravel()]).T
y += 0.5 - rng.rand(*y.shape)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=400, test_size=200, random_state=4)
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)

clf = DecisionTreeClassifier(random_state=0)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o",
drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
```

```
Text(0.5, 1.0, 'Total Impurity vs effective alpha for training set')
```


Total Impurity vs effective alpha for training set
total impurity of leaves
effective alpha
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035