I'm not a bot

Sklearn train and test split

Train Test Split Using Sklearn in Python ========== In this article, we will learn how to perform a train test split using the Sklearn library in Python. The `train_test_split()` method is used to divide our data into training and testing sets. ### How it Works First, we need to separate our data into features (X) and labels (y). We then use the `train_test_split()` function to create X_train, X_test, y_train, and y_test sets. The X_train and y_train sets are used for training and fitting the model, while the X_test and y_test sets are used for testing and validation. We can specify the size of the train and test sets using the `test_size` parameter. It is generally recommended to keep the train set larger than the test set. ### Syntax The syntax for the `train_test_split()` function is as follows: ```python sklearn.model_selection.train_test_split(X, y, test_size=None, train_size=None, random_state=None, shuffle=False, stratify=None) ``` Parameters: * `X`: The feature data. * `y`: The label data. * `test_size`: The percentage of the dataset to use for testing (float). * `train_size`: The percentage of the dataset to use for training (int or float). * `random_state`: The seed to use for random number generation (int). * `shuffle`: Whether to shuffle the data before splitting (bool). * `stratify`: The class labels to use for stratification (array-like object). ### Example Here is an example of how to use the `train_test_split()` function: ```python import numpy as np import pandas as pd from sklearn.model_selection import train_test_split # Load the dataset df = pd.read_csv('headbrain1.csv') # Separate the data into features (X) and labels (y) X = df['Head Size(cm^3)'] y = df['Brain Weight(grams)'] # Perform the train test split X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=104, test_size=0.25, shuffle=True) # Print the results print('X_train : ') print(X_train.head()) print('') print('X_test : ') print(X_test.head()) print('') print('y_train : ') print(y_train.head()) print('') print('y_test : ') print(y_test.head()) ``` This example uses a CSV file to load the dataset, and then separates the data into features (X) and labels (y). It then performs the train test split using the `train_test_split()` function, and prints the results. To split a Pandas DataFrame dataset into training and testing sets using the `train_test_split` function from scikit-learn, follow these steps. 1. First, import necessary libraries such as pandas and scikit-learn. 2. Then load your data into a DataFrame by specifying the path to your CSV file. 3. Next, prepare your features (X) and target variable (y) by dropping irrelevant columns and selecting only the relevant ones. 4. After that, use the `train_test_split` function to split your data into two parts: training set (X_train, y_train) and testing set (X_test, y_test). 5. The training set is used to train machine learning models, while the testing set is used to evaluate their performance. 6. By default, 25% of the data will be reserved for testing if not specified otherwise. 7. It's also a good practice to use a validation dataset (separate from both test and train datasets) to compare the performance of different candidate models. 8. The test set is used at the end to evaluate the selected model on completely new data, which minimizes overfitting. Here's an example: ```python import pandas as pd from sklearn.model_selection import train_test_split # Load your dataset into a DataFrame df = pd.read_csv('your_dataset.csv') # Prepare features and target variable X = df['Head Size(cm^3)'] y = df['Brain Weight(grams)'] # Split the data into training set (X_train, y_train) and testing set (X_test, y_test) X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=104, train_size=0.8, shuffle=True) print('Training Set:') print(X_train.head()) print(X_train.shape) print('\nTesting Set:') print(X_test.head()) print(X_test.shape) print('\nTarget Variable:') print(y_train.head()) print(y_train.shape) print('\nTarget Variable (Test):') print(y_test.head()) print(y_test.shape) ``` To transform data into a usable format, several steps are required. First, load the dataset into a pandas DataFrame using functions such as `pd.read_csv()` or `pd.read_excel()`. This step assumes that the dataset is already available; if not, create one using an example like the Iris dataset from scikit-learn. The Iris dataset consists of 150 samples from three species of flowers with four features measured from each sample: sepal and petal lengths and widths. To utilize this dataset in a Python script, import it into pandas and save it to a CSV file using `to_csv()`. The resulting DataFrame can be loaded into a Python environment for further processing. Next, prepare the features (independent variables) and target (dependent variable) by separating them into two distinct data structures. Then, use `train_test_split()` from scikit-learn to divide the dataset into training and test sets. The ratio of test data can be specified using the `test_size` argument, and a random seed can be set for reproducibility. Here's a step-by-step guide to preparing features and target (labels) and splitting the data: 1. **Import Required Libraries**: Ensure that necessary libraries like pandas and scikit-learn are installed in your Python environment. You can install them using pip if they haven't been installed already. 2. **Load Your Data into a DataFrame**: Import the required libraries and load your dataset into a pandas DataFrame using functions such as `pd.read_csv()` or `pd.read_excel()`. For example, you can use `pd.read_csv('iris_dataset.csv')` to load the Iris dataset saved earlier. 3. **Prepare Features and Target (Labels) and Split the Data**: Separate your features (independent variables) and target (dependent variable) into two different data structures. Then, use `train_test_split()` from scikit-learn to split the data into training and test sets. You can specify the ratio of test data with the `test_size` argument and set a random seed for reproducibility. 4. **Verify the Split**: Arguments of `train_test_split()` can be customized for advanced use, such as specifying different ratios for train and test sets or setting seeds for reproducibility. 5. **Split the Data Further**: Once the data is split into training and test sets, it may be necessary to further divide the data into a train, validation, and test set for model evaluation and hyperparameter tuning. This step can be achieved using techniques like stratified sampling or by manually specifying the ratios of each subset. By following these steps, you can efficiently prepare your dataset for machine learning tasks and ensure that it is split correctly to avoid any issues during model training and testing. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) The above code splits the dataset into training and test sets using the `train_test_split` function from Scikit-Learn. Here's how it works: * The `X` variable represents the features of the dataset, while the `y` variable represents the target or labels. * The `test_size=0.2` parameter means that 20% of the data will be used for the test set, and the rest (80%) will be used for the training set. * The `random_state=42` parameter ensures that the split is reproducible by setting a random seed. You can verify the split by checking the shape of the resulting dataframes: ```python print("Training set shapes:", X_train.shape, y_train.shape) print("Test set shapes:", X_test.shape, y_test.shape) ``` This will print the number of rows and columns in each dataframe. You can also check some datapoints to ensure that the split is correct: ```python print("Some features from the training set:") print(X_train) print("Some labels from the training set:") print(y_train) ``` The `train_test_split` function has several other parameters that you can use for advanced use cases. These include: * `arrays`: This parameter specifies the arrays to be split into train and test sets. * `test_size`: This parameter determines the proportion of the dataset that should be included in the test split. * `train_size`: This parameter determines the proportion of the dataset that should be included in the training split. * `random_state`: This parameter controls the shuffling applied to the data before applying the split. Here's how you can use these parameters: ```python # Using arrays X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # Using test_size and random_state X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Using train_size X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8) # Using shuffle X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True) # Using stratify from sklearn.utils import resample X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y) ``` Given text: function calls.Type: int, RandomState instance or None, default=NoneUsage: train_test_split(X, y, random_state=42)5. What is the shuffle parameter and how to use it?shuffle determines whether or not to shuffle the data before splitting.Type: bool, default=TrueDescription: Whether or not to shuffle the data before splitting.Usage: train_test_split(X, y, shuffle=False) to avoid shuffling.6. What is the stratify parameter and how to use it? In short, stratify ensures that the data is split in a stratified fashion, using the provided array as class labels.“Stratify” refers to ensuring that the proportions of different classes (or outcomes) in the dataset are maintained consistently across both training and test sets when splitting the data. This is particularly important for imbalanced datasets to ensure that all classes are adequately represented in both subsets. Here is a helpful diagram of what this means: Type: array-like or None, default=None Usage: train_test_split(X, y, stratify=y) to ensure the same distribution of classes in both train and test sets. Example usage with all the arguments: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=42, shuffle=True, stratify=y) In the above example, 80% of the data is used for training, 20% for testing, the data is shuffled before splitting, and the split is stratified based on the class labels in y. The random_state ensures that the split is reproducible. Remember that test_size, train_size, random_state, shuffle, and stratify are optional parameters. If test_size is None, the value is set to the complement of train_size. Given text: "data goes to the validation set, and 20% goes to the test set.Step 4: Verify the SplitsYou might want to verify the splits by checking the shapes of the resulting datasets.print("Training set:", X_train.shape, y_train.shape)print("Validation set:", X_val.shape, y_val.shape)print("Test set:", X_test.shape, y_test.shape)If you follow my example from before, and replace the 'your_dataset.csv' from step 1 with 'iris_dataset.csv' and run this code, you should get the following output:Training set: (90, 4) (90,)Validation set: (30, 4) (30,)Test set: (30, 4) (30,)Now you have successfully split your dataset into training (60%), validation (20%), and test (20%) sets. Identifying which category an object belongs to. Applications: Spam detection, image recognition. Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more... Page 2 For a short description of the main highlights of the release, please refer to Release Highlights for scikit-learn 1.6. Legend for changelogs Major Feature something big that you couldn’t do before. Feature something that you couldn’t do before. Efficiency an existing feature now may not require as much computation or memory. Enhancement a miscellaneous minor improvement. Fix something that previously didn’t work as documented – or according to reasonable expectations – should now work. API Change you will need to change your code to have the same effect in the future; or a feature will be removed in the future. January 2025 Fix The tags.input_tags.sparse flag was corrected for a majority of estimators. By Antoine Baker #30187 Fix Fix regression when scikit-learn metric called on PyTorch CPU tensors would raise an error (with array API dispatch disabled which is the default). By Loïc Estève #30454 Fix Use log2 instead of ln for building trees to maintain behavior of previous versions. By Thomas Fan #30557 Enhancement utils.estimator_checks.check_estimator_sparse_tag ensures that the estimator tag input_tags.sparse is consistent with its fit method (accepting sparse input X or raising the appropriate error). By Antoine Baker #30187 Fix Raise a DeprecationWarning when there is no concrete implementation of __sklearn_tags__ in the MRO of the estimator. We request to inherit from BaseEstimator that implements __sklearn_tags__. By Guillaume Lemaitre #30516 December 2024 Additional estimators and functions have been updated to include support for all Array API compliant inputs. See Array API support (experimental) for more details. Feature model_selection.GridSearchCV, model_selection.RandomizedSearchCV, model_selection.HalvingGridSearchCV and model_selection.HalvingRandomSearchCV now support Array API compatible inputs when their base estimators do. By Tim Head and Olivier Grisel #27096 Feature sklearn.metrics.f1_score now supports Array API compatible inputs. By Omar Salman #27369 Feature preprocessing.LabelEncoder now supports Array API compatible inputs. By Omar Salman #27381 Feature sklearn.metrics.mean_absolute_error now supports Array API compatible inputs. By Edoardo Abati #27736 Feature sklearn.metrics.mean_tweedie_deviance now supports Array API compatible inputs. By Thomas Li #28106 Feature sklearn.metrics.pairwise.cosine_similarity now supports Array API compatible inputs. By Edoardo Abati Feature updates for sklearn.metrics include support for Array API compatible inputs for paired_cosine_distances, entropy, mean_squared_error, additive_chi2_kernel, d2_tweedie_score, max_error, mean_poisson_deviance, mean_gamma_deviance, cosine_distances, chi2_kernel, paired_euclidean_distances, euclidean_distances, rbf_kernel, linear_kernel, sigmoid_kernel, polynomial_kernel, and mean_squared_log_error. Additionally, preprocessing.MinMaxScaler with clip=True now supports Array API compatible inputs. The cupy.array_api module has been removed in favor of directly supporting the top-level cupy module, possibly via the array_api_compat.cupy compatibility wrapper. Official PyPy support has also been dropped due to limited maintainer resources and a small number of users. scikit-learn now has preliminary support for free-threaded CPython, which aims to enable efficient multi-threaded use cases by removing the Global Interpreter Lock (GIL). This is an experimental version of CPython 3.13, and it can be installed with free-threaded wheels available for all supported platforms. The copy parameter of cluster.Birch was deprecated in 1.6 and will be removed in 1.8. By Yao Xiao, Olivier Grisel, and others #Various Feature The datasets.fetch_file function allows downloading arbitrary data files from the web with local caching, integrity checks using SHA256 digests, and automatic retries in case of HTTP errors. Additionally, the FrozenEstimator is introduced to freeze an estimator, preventing it from performing in-place operations on the input data. Furthermore, the solver="newton-cholesky" in linear_model.LogisticRegression and linear_model.LogisticRegressionCV now supports the full multinomial loss in a multiclass setting. #Various Fix In linear_model.Ridge and linear_model.RidgeCV, after fitting, the coef_ attribute is now of shape (n_samples,) like other linear models. The sample weight handling for test scores has been corrected in linear_model.LogisticRegressionCV. Also, linear_model.LassoCV and linear_model.ElasticNetCV now consider sample weights when defining the search grid for the internally tuned alpha hyper-parameter. Furthermore, linear_model.LogisticRegression, linear_model.PoissonRegressor, linear_model.GammaRegressor, and linear_model.TweedieRegressor now take sample weights into account to decide when to fall back to solver='lbfgs'. Additionally, linear_model.RidgeCV now properly uses predictions on the same scale as the target seen during fit. It also supports custom multioutput scorers by letting the scorer manage the multioutput averaging. Finally, linear_model.LinearRegression sets the cond parameter when calling the scipy.linalg.lstsq solver on dense input data to ensure more numerically robust results on rank-deficient data. #Various Fix Given article text here Looking forward to seeing everyone at the meeting tomorrow and discussing our strategies. By Yao Xiao #26367 Enhancement sklearn.metrics.check_scoring now accepts raise_exc to specify whether to raise an exception if a subset of the scorers in multimetric scoring fails or to return an error code. By Stefanie Senger #28992 Fix metrics.roc_auc_score will now correctly return np.nan and warn user if only one class is present in the labels. By Gleb Levitski and Janez Demšar #27412, #30013 Fix The functions metrics.mean_squared_log_error and metrics.root_mean_squared_log_error now check whether the inputs are within the correct domain for the function \(y=\log(1+x)\), rather than \(y=\log(x)\). The functions metrics.mean_absolute_error, metrics.mean_absolute_percentage_error, metrics.mean_squared_error and metrics.root_mean_squared_error now explicitly check whether a scalar will be returned when multioutput=uniform_average. By Virgil Chan #29709 API Change The assert_all_finite parameter of functions metrics.pairwise.check_pairwise_arrays and metrics.pairwise_distances is renamed into ensure_allFinite. force_allFinite will be removed in 1.8. By Jérémie du Boisberranger #29404 API Change scoring="neg_max_error" should be used instead of scoring="max_error" which is now deprecated. By Farid “Freddie” Taba #29462 API Change The default value of the response_method parameter of metrics.make_scorer will change from None to "predict" and None will be removed in 1.8. In the mean time, None is equivalent to "predict". By Jérémie du Boisberranger #30001 Enhancement neighbors.NearestNeighbors, neighbors.KNeighborsClassifier, neighbors.KNeighborsRegressor, neighbors.RadiusNeighborsClassifier, neighbors.RadiusNeighborsRegressor, neighbors.KNeighborsTransformer, neighbors.RadiusNeighborsTransformer, and neighbors.LocalOutlierFactor now work with metric="nan_euclidean", supporting nan inputs. By Carlo Lemos, Guillaume Lemaitre, and Adrin Jalali #25330 Enhancement Add neighbors.NearestCentroid.decision_function, neighbors.NearestCentroid.predict_proba and neighbors.NearestCentroid.predict_log_proba to the neighbors.NearestCentroid estimator class. Support the case when X is sparse and shrinking_threshold is not None in neighbors.NearestCentroid. By Matthew Ning #26689 Enhancement Make predict, predict_proba, and score of neighbors.KNeighborsClassifier and neighbors.RadiusNeighborsClassifier accept X=None as input. In this case predictions for all training set points are returned, and points are not included into their own neighbors. By Dmitry Kobak #30047 Fix neighbors.LocalOutlierFactor raises a warning in the fit method when duplicate values in the training data lead to inaccurate outlier detection. By Henrique Caroço #28773 Major Feature pipeline.Pipeline can now transform metadata up to the step requiring the metadata, which can be set using the transform_input parameter. By Adrin Jalali #28901 Enhancement pipeline.Pipeline now warns about not being fitted before calling methods that require the pipeline to be fitted. This warning will become an error in 1.8. By Adrin Jalali #29868 Fix Fixed an issue with Pipeline estimator type handling has been improved when the pipeline is empty. This enables correct rendering of HTML representations for empty pipelines. (Gennaro Daniele Acciaro #30203) The `utils.check_array` function now accepts an additional parameter, `ensure_non_negative`, to check for negative values in arrays. Previously, this feature was only accessible through calling `utils.check_non_negative`. (Tamara Atanasoska #29540) Changes have been made to the behavior of estimators and their associated tags. Specifically, the `check_estimator` function now fails if a classifier is tagged as multi-class but does not support it, while previously it did not check for this condition on multi-class data. (Adrin Jalali #29874, #29880) The `utils.validation.check_is_fitted` function has been updated to handle stateless estimators correctly. Statelessness can be indicated by setting the `requires_fit` tag. See Estimator Tags for more information. (Adrin Jalali #29880) Additional enhancements and bug fixes have been made, including: - Support for estimators with set_output methods in `parametrize_with_checks` and `check_estimator`. (Adrin Jalali #30149) - Renaming of the `assert_all_finite` parameter to `ensure_all_finite` in functions like `utils.check_array`. The deprecated `force_all_finite` will be removed in version 1.8. (Jérémie du Boisberranger #29404) - Replacement of `check_sample_weights_invariance` with `check_sample_weight_equivalence_on_dense_data` and `check_sample_weight_equivalence_on_sparse_data`, which handle integer weights including zero, on dense and sparse data respectively. (Antoine Baker #29818, #30137) - Deprecation of setting estimator type through the `_estimator_type` attribute in favor of inheritance from mixin classes or using the `estimator_type` tag in the `__sklearn_tags__` method. (Adrin Jalali #30122) The project's contributors since version 1.5 have been acknowledged, including several individuals and organizations that have contributed to its maintenance and improvement. The Scikit-Learn library is an open-source Python module for machine learning that has been built on top of SciPy since its inception in 2007 by David Cournapeau as part of Google Summer of Code project. Over the years, numerous volunteers have contributed to the project's development, with a current team of maintainers working together to ensure its continued growth and improvement. Scikit-Learn requires Python version 3.10 or higher, NumPy version 1.22.0 or higher, SciPy version 1.8.0 or higher, joblib version 1.2.0 or higher, threadpoolctl version 3.1.0 or higher, and Matplotlib version 3.5.0 or higher to function properly. Certain features also require additional packages such as scikit-image, pandas, seaborn, and plotly. The easiest way to install Scikit-Learn is through pip: `pip install -U scikit-learn` or conda: `conda install -c conda-forge scikit-learn` More detailed installation instructions can be found in the documentation. The project's changelog provides a history of notable changes and updates. Scikit-Learn welcomes new contributors from all backgrounds and experience levels, striving to maintain a culture that is helpful, welcoming, and effective. Guide for contributors includes detailed info on code contrib, documentation, tests and more. Basic info is included in this README. Check the latest sources with git clone To learn more about contrib, see our Contributing guide. After installation, you can run pytest sklearn. See for testing info. Environment variable SKLEARN_SEED controlls random number generation during testing. Before opening PR, check the full Contributing page: The project was started in 2007 by David Cournapeau as a Google Summer of Code project. Many volunteers have contrib since then. See About us page for core contributors.