**Python API Reference**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#python-api-reference)

This page gives the Python API reference of xgboost, please also refer to Python Package Introduction for more information about the Python package.

* [Global Configuration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#global-configuration)
* [Core Data Structure](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.core)
* [Learning API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.training)
* [Scikit-Learn API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.sklearn)
* [Plotting API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.plotting)
* [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.callback)
* [Dask API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.dask)
  + [Dask extensions for distributed training](https://xgboost.readthedocs.io/en/stable/python/python_api.html#dask-extensions-for-distributed-training)
    - [Optional dask configuration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#optional-dask-configuration)

**Global Configuration**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#global-configuration)

xgboost.config\_context(*\*\*new\_config*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.config_context)

Context manager for global XGBoost configuration.

Global configuration consists of a collection of parameters that can be applied in the global scope. See [Global Configuration](https://xgboost.readthedocs.io/en/stable/parameter.html#global-config) for the full list of parameters supported in the global configuration.

Note

All settings, not just those presently modified, will be returned to their previous values when the context manager is exited. This is not thread-safe.

New in version 1.4.0.

Parameters

**new\_config** (*Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, Any]*) – Keyword arguments representing the parameters and their values

Example

import xgboost as xgb

# Show all messages, including ones pertaining to debugging

xgb.set\_config(verbosity=2)

# Get current value of global configuration

# This is a dict containing all parameters in the global configuration,

# including 'verbosity'

config = xgb.get\_config()

assert config['verbosity'] == 2

# Example of using the context manager xgb.config\_context().

# The context manager will restore the previous value of the global

# configuration upon exiting.

with xgb.config\_context(verbosity=0):

# Suppress warning caused by model generated with XGBoost version < 1.0.0

bst = xgb.Booster(model\_file='./old\_model.bin')

assert xgb.get\_config()['verbosity'] == 2 # old value restored

See also

[set\_config](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.set_config)

Set global XGBoost configuration

[get\_config](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.get_config)

Get current values of the global configuration

xgboost.set\_config(*\*\*new\_config*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.set_config)

Set global configuration.

Global configuration consists of a collection of parameters that can be applied in the global scope. See [Global Configuration](https://xgboost.readthedocs.io/en/stable/parameter.html#global-config) for the full list of parameters supported in the global configuration.

New in version 1.4.0.

Parameters

**new\_config** (*Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, Any]*) – Keyword arguments representing the parameters and their values

Example

import xgboost as xgb

# Show all messages, including ones pertaining to debugging

xgb.set\_config(verbosity=2)

# Get current value of global configuration

# This is a dict containing all parameters in the global configuration,

# including 'verbosity'

config = xgb.get\_config()

assert config['verbosity'] == 2

# Example of using the context manager xgb.config\_context().

# The context manager will restore the previous value of the global

# configuration upon exiting.

with xgb.config\_context(verbosity=0):

# Suppress warning caused by model generated with XGBoost version < 1.0.0

bst = xgb.Booster(model\_file='./old\_model.bin')

assert xgb.get\_config()['verbosity'] == 2 # old value restored

xgboost.get\_config()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.get_config)

Get current values of the global configuration.

Global configuration consists of a collection of parameters that can be applied in the global scope. See [Global Configuration](https://xgboost.readthedocs.io/en/stable/parameter.html#global-config) for the full list of parameters supported in the global configuration.

New in version 1.4.0.

Returns

**args** – The list of global parameters and their values

Return type

Dict[[str](https://docs.python.org/3.6/library/stdtypes.html#str), Any]

Example

import xgboost as xgb

# Show all messages, including ones pertaining to debugging

xgb.set\_config(verbosity=2)

# Get current value of global configuration

# This is a dict containing all parameters in the global configuration,

# including 'verbosity'

config = xgb.get\_config()

assert config['verbosity'] == 2

# Example of using the context manager xgb.config\_context().

# The context manager will restore the previous value of the global

# configuration upon exiting.

with xgb.config\_context(verbosity=0):

# Suppress warning caused by model generated with XGBoost version < 1.0.0

bst = xgb.Booster(model\_file='./old\_model.bin')

assert xgb.get\_config()['verbosity'] == 2 # old value restored

**Core Data Structure**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.core)

Core XGBoost Library.

*class* xgboost.DMatrix(*data*, *label=None*, *\**, *weight=None*, *base\_margin=None*, *missing=None*, *silent=False*, *feature\_names=None*, *feature\_types=None*, *nthread=None*, *group=None*, *qid=None*, *label\_lower\_bound=None*, *label\_upper\_bound=None*, *feature\_weights=None*, *enable\_categorical=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)

Bases: [object](https://docs.python.org/3.6/library/functions.html#object)

Data Matrix used in XGBoost.

DMatrix is an internal data structure that is used by XGBoost, which is optimized for both memory efficiency and training speed. You can construct DMatrix from multiple different sources of data.

Parameters

* **data** (*os.PathLike/string/numpy.array/scipy.sparse/pd.DataFrame/*) – dt.Frame/cudf.DataFrame/cupy.array/dlpack Data source of DMatrix. When data is string or os.PathLike type, it represents the path libsvm format txt file, csv file (by specifying uri parameter ‘path\_to\_csv?format=csv’), or binary file that xgboost can read from.
* **label** (*array\_like*) – Label of the training data.
* **weight** (*array\_like*) –

Weight for each instance.

Note

For ranking task, weights are per-group.

In ranking task, one weight is assigned to each group (not each data point). This is because we only care about the relative ordering of data points within each group, so it doesn’t make sense to assign weights to individual data points.

* **base\_margin** (*array\_like*) – Base margin used for boosting from existing model.
* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, optional*) – Value in the input data which needs to be present as a missing value. If None, defaults to np.nan.
* **silent** (*boolean, optional*) – Whether print messages during construction
* **feature\_names** ([*list*](https://docs.python.org/3.6/library/stdtypes.html#list)*, optional*) – Set names for features.
* **feature\_types** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Set types for features. When *enable\_categorical* is set to *True*, string “c” represents categorical data type.
* **nthread** (*integer, optional*) – Number of threads to use for loading data when parallelization is applicable. If -1, uses maximum threads available on the system.
* **group** (*array\_like*) – Group size for all ranking group.
* **qid** (*array\_like*) – Query ID for data samples, used for ranking.
* **label\_lower\_bound** (*array\_like*) – Lower bound for survival training.
* **label\_upper\_bound** (*array\_like*) – Upper bound for survival training.
* **feature\_weights** (*array\_like, optional*) – Set feature weights for column sampling.
* **enable\_categorical** (*boolean, optional*) –

New in version 1.3.0.

Note

This parameter is experimental

Experimental support of specializing for categorical features. Do not set to True unless you are interested in development. Also, JSON/UBJSON serialization format is required.

Return type

None

*property* feature\_names*: Optional[List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.feature_names)

Get feature names (column labels).

Returns

**feature\_names**

Return type

[list](https://docs.python.org/3.6/library/stdtypes.html#list) or None

*property* feature\_types*: Optional[List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.feature_types)

Get feature types (column types).

Returns

**feature\_types**

Return type

[list](https://docs.python.org/3.6/library/stdtypes.html#list) or None

get\_base\_margin()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.get_base_margin)

Get the base margin of the DMatrix.

Return type

base\_margin

get\_float\_info(*field*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.get_float_info)

Get float property from the DMatrix.

Parameters

**field** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – The field name of the information

Returns

**info** – a numpy array of float information of the data

Return type

array

get\_group()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.get_group)

Get the group of the DMatrix.

Return type

group

get\_label()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.get_label)

Get the label of the DMatrix.

Returns

**label**

Return type

array

get\_uint\_info(*field*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.get_uint_info)

Get unsigned integer property from the DMatrix.

Parameters

**field** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – The field name of the information

Returns

**info** – a numpy array of unsigned integer information of the data

Return type

array

get\_weight()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.get_weight)

Get the weight of the DMatrix.

Returns

**weight**

Return type

array

num\_col()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.num_col)

Get the number of columns (features) in the DMatrix.

Returns

**number of columns**

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

num\_row()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.num_row)

Get the number of rows in the DMatrix.

Returns

**number of rows**

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

save\_binary(*fname*, *silent=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.save_binary)

Save DMatrix to an XGBoost buffer. Saved binary can be later loaded by providing the path to [xgboost.DMatrix()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix) as input.

Parameters

* **fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Name of the output buffer file.
* **silent** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool) *(optional; default: True)*) – If set, the output is suppressed.

Return type

None

set\_base\_margin(*margin*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.set_base_margin)

Set base margin of booster to start from.

This can be used to specify a prediction value of existing model to be base\_margin However, remember margin is needed, instead of transformed prediction e.g. for logistic regression: need to put in value before logistic transformation see also example/demo.py

Parameters

**margin** (*array like*) – Prediction margin of each datapoint

Return type

None

set\_float\_info(*field*, *data*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.set_float_info)

Set float type property into the DMatrix.

Parameters

* **field** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – The field name of the information
* **data** (*numpy array*) – The array of data to be set

Return type

None

set\_float\_info\_npy2d(*field*, *data*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.set_float_info_npy2d)

Set float type property into the DMatrix

for numpy 2d array input

Parameters

* **field** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – The field name of the information
* **data** (*numpy array*) – The array of data to be set

Return type

None

set\_group(*group*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.set_group)

Set group size of DMatrix (used for ranking).

Parameters

**group** (*array like*) – Group size of each group

Return type

None

set\_info(*\**, *label=None*, *weight=None*, *base\_margin=None*, *group=None*, *qid=None*, *label\_lower\_bound=None*, *label\_upper\_bound=None*, *feature\_names=None*, *feature\_types=None*, *feature\_weights=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.set_info)

Set meta info for DMatrix. See doc string for [xgboost.DMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix).

Parameters

* **label** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –
* **weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –
* **group** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –
* **qid** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –
* **label\_lower\_bound** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –
* **label\_upper\_bound** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –
* **feature\_names** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) –
* **feature\_types** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) –
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –

Return type

None

set\_label(*label*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.set_label)

Set label of dmatrix

Parameters

**label** (*array like*) – The label information to be set into DMatrix

Return type

None

set\_uint\_info(*field*, *data*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.set_uint_info)

Set uint type property into the DMatrix.

Parameters

* **field** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – The field name of the information
* **data** (*numpy array*) – The array of data to be set

Return type

None

set\_weight(*weight*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.set_weight)

Set weight of each instance.

Parameters

**weight** (*array like*) –

Weight for each data point

Note

For ranking task, weights are per-group.

In ranking task, one weight is assigned to each group (not each data point). This is because we only care about the relative ordering of data points within each group, so it doesn’t make sense to assign weights to individual data points.

Return type

None

slice(*rindex*, *allow\_groups=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix.slice)

Slice the DMatrix and return a new DMatrix that only contains *rindex*.

Parameters

* **rindex** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]*) – List of indices to be selected.
* **allow\_groups** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Allow slicing of a matrix with a groups attribute

Returns

A new DMatrix containing only selected indices.

Return type

res

*class* xgboost.DeviceQuantileDMatrix(*data*, *label=None*, *\**, *weight=None*, *base\_margin=None*, *missing=None*, *silent=False*, *feature\_names=None*, *feature\_types=None*, *nthread=None*, *max\_bin=256*, *group=None*, *qid=None*, *label\_lower\_bound=None*, *label\_upper\_bound=None*, *feature\_weights=None*, *enable\_categorical=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DeviceQuantileDMatrix)

Bases: [xgboost.core.DMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)

Device memory Data Matrix used in XGBoost for training with tree\_method=’gpu\_hist’. Do not use this for test/validation tasks as some information may be lost in quantisation. This DMatrix is primarily designed to save memory in training from device memory inputs by avoiding intermediate storage. Set max\_bin to control the number of bins during quantisation. See doc string in [xgboost.DMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix) for documents on meta info.

You can construct DeviceQuantileDMatrix from cupy/cudf/dlpack.

New in version 1.1.0.

Parameters

* **data** (*os.PathLike/string/numpy.array/scipy.sparse/pd.DataFrame/*) – dt.Frame/cudf.DataFrame/cupy.array/dlpack Data source of DMatrix. When data is string or os.PathLike type, it represents the path libsvm format txt file, csv file (by specifying uri parameter ‘path\_to\_csv?format=csv’), or binary file that xgboost can read from.
* **label** (*array\_like*) – Label of the training data.
* **weight** (*array\_like*) –

Weight for each instance.

Note

For ranking task, weights are per-group.

In ranking task, one weight is assigned to each group (not each data point). This is because we only care about the relative ordering of data points within each group, so it doesn’t make sense to assign weights to individual data points.

* **base\_margin** (*array\_like*) – Base margin used for boosting from existing model.
* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, optional*) – Value in the input data which needs to be present as a missing value. If None, defaults to np.nan.
* **silent** (*boolean, optional*) – Whether print messages during construction
* **feature\_names** ([*list*](https://docs.python.org/3.6/library/stdtypes.html#list)*, optional*) – Set names for features.
* **feature\_types** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Set types for features. When *enable\_categorical* is set to *True*, string “c” represents categorical data type.
* **nthread** (*integer, optional*) – Number of threads to use for loading data when parallelization is applicable. If -1, uses maximum threads available on the system.
* **group** (*array\_like*) – Group size for all ranking group.
* **qid** (*array\_like*) – Query ID for data samples, used for ranking.
* **label\_lower\_bound** (*array\_like*) – Lower bound for survival training.
* **label\_upper\_bound** (*array\_like*) – Upper bound for survival training.
* **feature\_weights** (*array\_like, optional*) – Set feature weights for column sampling.
* **enable\_categorical** (*boolean, optional*) –

New in version 1.3.0.

Note

This parameter is experimental

Experimental support of specializing for categorical features. Do not set to True unless you are interested in development. Also, JSON/UBJSON serialization format is required.

* **max\_bin** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Return type

None

*class* xgboost.Booster(*params=None*, *cache=None*, *model\_file=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)

Bases: [object](https://docs.python.org/3.6/library/functions.html#object)

A Booster of XGBoost.

Booster is the model of xgboost, that contains low level routines for training, prediction and evaluation.

Parameters

* **params** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)) – Parameters for boosters.
* **cache** ([*list*](https://docs.python.org/3.6/library/stdtypes.html#list)) – List of cache items.
* **model\_file** (*string/os.PathLike/Booster/bytearray*) – Path to the model file if it’s string or PathLike.

Return type

None

attr(*key*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.attr)

Get attribute string from the Booster.

Parameters

**key** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – The key to get attribute from.

Returns

**value** – The attribute value of the key, returns None if attribute do not exist.

Return type

[str](https://docs.python.org/3.6/library/stdtypes.html#str)

attributes()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.attributes)

Get attributes stored in the Booster as a dictionary.

Returns

**result** – Returns an empty dict if there’s no attributes.

Return type

dictionary of attribute\_name: attribute\_value pairs of strings.

boost(*dtrain*, *grad*, *hess*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.boost)

Boost the booster for one iteration, with customized gradient statistics. Like [xgboost.Booster.update()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.update), this function should not be called directly by users.

Parameters

* **dtrain** ([*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)) – The training DMatrix.
* **grad** ([*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)) – The first order of gradient.
* **hess** ([*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)) – The second order of gradient.

Return type

None

copy()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.copy)

Copy the booster object.

Returns

**booster** – a copied booster model

Return type

*Booster*

dump\_model(*fout*, *fmap=''*, *with\_stats=False*, *dump\_format='text'*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.dump_model)

Dump model into a text or JSON file. Unlike [save\_model()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.save_model), the output format is primarily used for visualization or interpretation, hence it’s more human readable but cannot be loaded back to XGBoost.

Parameters

* **fout** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name.
* **fmap** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*, optional*) – Name of the file containing feature map names.
* **with\_stats** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)*, optional*) – Controls whether the split statistics are output.
* **dump\_format** (*string, optional*) – Format of model dump file. Can be ‘text’ or ‘json’.

Return type

None

eval(*data*, *name='eval'*, *iteration=0*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.eval)

Evaluate the model on mat.

Parameters

* **data** ([*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)) – The dmatrix storing the input.
* **name** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – The name of the dataset.
* **iteration** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – The current iteration number.

Returns

**result** – Evaluation result string.

Return type

[str](https://docs.python.org/3.6/library/stdtypes.html#str)

eval\_set(*evals*, *iteration=0*, *feval=None*, *output\_margin=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.eval_set)

Evaluate a set of data.

Parameters

* **evals** ([*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*,* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – List of items to be evaluated.
* **iteration** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Current iteration.
* **feval** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]*) – Custom evaluation function.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Returns

**result** – Evaluation result string.

Return type

[str](https://docs.python.org/3.6/library/stdtypes.html#str)

*property* feature\_names*: Optional[List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.feature_names)

Feature names for this booster. Can be directly set by input data or by assignment.

*property* feature\_types*: Optional[List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.feature_types)

Feature types for this booster. Can be directly set by input data or by assignment.

get\_dump(*fmap=''*, *with\_stats=False*, *dump\_format='text'*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.get_dump)

Returns the model dump as a list of strings. Unlike [save\_model()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.save_model), the output format is primarily used for visualization or interpretation, hence it’s more human readable but cannot be loaded back to XGBoost.

Parameters

* **fmap** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Name of the file containing feature map names.
* **with\_stats** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Controls whether the split statistics are output.
* **dump\_format** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – Format of model dump. Can be ‘text’, ‘json’ or ‘dot’.

Return type

[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)[[str](https://docs.python.org/3.6/library/stdtypes.html#str)]

get\_fscore(*fmap=''*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.get_fscore)

Get feature importance of each feature.

Note

Zero-importance features will not be included

Keep in mind that this function does not include zero-importance feature, i.e. those features that have not been used in any split conditions.

Parameters

**fmap** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – The name of feature map file

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)[[float](https://docs.python.org/3.6/library/functions.html#float), [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)[[float](https://docs.python.org/3.6/library/functions.html#float)]]]

get\_score(*fmap=''*, *importance\_type='weight'*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.get_score)

Get feature importance of each feature. For tree model Importance type can be defined as:

* ‘weight’: the number of times a feature is used to split the data across all trees.
* ‘gain’: the average gain across all splits the feature is used in.
* ‘cover’: the average coverage across all splits the feature is used in.
* ‘total\_gain’: the total gain across all splits the feature is used in.
* ‘total\_cover’: the total coverage across all splits the feature is used in.

Note

For linear model, only “weight” is defined and it’s the normalized coefficients without bias.

Note

Zero-importance features will not be included

Keep in mind that this function does not include zero-importance feature, i.e. those features that have not been used in any split conditions.

Parameters

* **fmap** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – The name of feature map file.
* **importance\_type** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – One of the importance types defined above.

Returns

* A map between feature names and their scores. When *gblinear* is used for
* *multi-class classification the scores for each feature is a list with length*
* *n\_classes*, otherwise they’re scalars.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)[[float](https://docs.python.org/3.6/library/functions.html#float), [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)[[float](https://docs.python.org/3.6/library/functions.html#float)]]]

get\_split\_value\_histogram(*feature*, *fmap=''*, *bins=None*, *as\_pandas=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.get_split_value_histogram)

Get split value histogram of a feature

Parameters

* **feature** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – The name of the feature.
* **fmap** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str) *or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike) *(optional)*) – The name of feature map file.
* **bin** ([*int*](https://docs.python.org/3.6/library/functions.html#int)*, default None*) – The maximum number of bins. Number of bins equals number of unique split values n\_unique, if bins == None or bins > n\_unique.
* **as\_pandas** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)*, default True*) – Return pd.DataFrame when pandas is installed. If False or pandas is not installed, return numpy ndarray.
* **bins** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

Returns

* *a histogram of used splitting values for the specified feature*
* *either as numpy array or pandas DataFrame.*

Return type

[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)[[numpy.ndarray](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray), pandas.core.frame.DataFrame]

inplace\_predict(*data*, *iteration\_range=(0, 0)*, *predict\_type='value'*, *missing=nan*, *validate\_features=True*, *base\_margin=None*, *strict\_shape=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.inplace_predict)

Run prediction in-place, Unlike [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.predict) method, inplace prediction does not cache the prediction result.

Calling only inplace\_predict in multiple threads is safe and lock free. But the safety does not hold when used in conjunction with other methods. E.g. you can’t train the booster in one thread and perform prediction in the other.

booster.set\_param({'predictor': 'gpu\_predictor'})

booster.inplace\_predict(cupy\_array)

booster.set\_param({'predictor': 'cpu\_predictor})

booster.inplace\_predict(numpy\_array)

New in version 1.1.0.

Parameters

* **data** (*numpy.ndarray/scipy.sparse.csr\_matrix/cupy.ndarray/*) – cudf.DataFrame/pd.DataFrame The input data, must not be a view for numpy array. Set predictor to gpu\_predictor for running prediction on CuPy array or CuDF DataFrame.
* **iteration\_range** ([*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.predict) for details.
* **predict\_type** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) –
  + *value* Output model prediction values.
  + *margin* Output the raw untransformed margin value.
* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)) – See [xgboost.DMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix) for details.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – See [xgboost.Booster.predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.predict) for details.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –

See [xgboost.DMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix) for details.

New in version 1.4.0.

* **strict\_shape** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

See [xgboost.Booster.predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.predict) for details.

New in version 1.4.0.

Returns

**prediction** – The prediction result. When input data is on GPU, prediction result is stored in a cupy array.

Return type

numpy.ndarray/cupy.ndarray

load\_config(*config*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.load_config)

Load configuration returned by *save\_config*.

New in version 1.0.0.

Parameters

**config** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) –

Return type

None

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

num\_boosted\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.num_boosted_rounds)

Get number of boosted rounds. For gblinear this is reset to 0 after serializing the model.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

num\_features()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.num_features)

Number of features in booster.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

predict(*data*, *output\_margin=False*, *ntree\_limit=0*, *pred\_leaf=False*, *pred\_contribs=False*, *approx\_contribs=False*, *pred\_interactions=False*, *validate\_features=True*, *training=False*, *iteration\_range=(0, 0)*, *strict\_shape=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.predict)

Predict with data. The full model will be used unless *iteration\_range* is specified, meaning user have to either slice the model or use the best\_iteration attribute to get prediction from best model returned from early stopping.

Note

See [Prediction](https://xgboost.readthedocs.io/en/stable/prediction.html) for issues like thread safety and a summary of outputs from this function.

Parameters

* **data** ([*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)) – The dmatrix storing the input.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use *iteration\_range* instead.
* **pred\_leaf** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this option is on, the output will be a matrix of (nsample, ntrees) with each record indicating the predicted leaf index of each sample in each tree. Note that the leaf index of a tree is unique per tree, so you may find leaf 1 in both tree 1 and tree 0.
* **pred\_contribs** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True the output will be a matrix of size (nsample, nfeats + 1) with each record indicating the feature contributions (SHAP values) for that prediction. The sum of all feature contributions is equal to the raw untransformed margin value of the prediction. Note the final column is the bias term.
* **approx\_contribs** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Approximate the contributions of each feature. Used when pred\_contribs or pred\_interactions is set to True. Changing the default of this parameter (False) is not recommended.
* **pred\_interactions** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True the output will be a matrix of size (nsample, nfeats + 1, nfeats + 1) indicating the SHAP interaction values for each pair of features. The sum of each row (or column) of the interaction values equals the corresponding SHAP value (from pred\_contribs), and the sum of the entire matrix equals the raw untransformed margin value of the prediction. Note the last row and column correspond to the bias term.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **training** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Whether the prediction value is used for training. This can effect *dart* booster, which performs dropouts during training iterations but use all trees for inference. If you want to obtain result with dropouts, set this parameter to *True*. Also, the parameter is set to true when obtaining prediction for custom objective function.

New in version 1.0.0.

* **iteration\_range** ([*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying *iteration\_range=(10, 20)*, then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

* **strict\_shape** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

When set to True, output shape is invariant to whether classification is used. For both value and margin prediction, the output shape is (n\_samples, n\_groups), n\_groups == 1 when multi-class is not used. Default to False, in which case the output shape can be (n\_samples, ) if multi-class is not used.

New in version 1.4.0.

Returns

**prediction**

Return type

numpy array

save\_config()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.save_config)

Output internal parameter configuration of Booster as a JSON string.

New in version 1.0.0.

Return type

[str](https://docs.python.org/3.6/library/stdtypes.html#str)

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

save\_raw(*raw\_format='deprecated'*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.save_raw)

Save the model to a in memory buffer representation instead of file.

Parameters

**raw\_format** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – Format of output buffer. Can be *json*, *ubj* or *deprecated*. Right now the default is *deprecated* but it will be changed to *ubj* (univeral binary json) in the future.

Return type

An in memory buffer representation of the model

set\_attr(*\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.set_attr)

Set the attribute of the Booster.

Parameters

* **\*\*kwargs** – The attributes to set. Setting a value to None deletes an attribute.
* **kwargs** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

Return type

None

set\_param(*params*, *value=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.set_param)

Set parameters into the Booster.

Parameters

* **params** (*dict/list/str*) – list of key,value pairs, dict of key to value or simply str key
* **value** (*optional*) – value of the specified parameter, when params is str key

Return type

None

trees\_to\_dataframe(*fmap=''*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.trees_to_dataframe)

Parse a boosted tree model text dump into a pandas DataFrame structure.

This feature is only defined when the decision tree model is chosen as base learner (*booster in {gbtree, dart}*). It is not defined for other base learner types, such as linear learners (*booster=gblinear*).

Parameters

**fmap** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str) *or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike) *(optional)*) – The name of feature map file.

Return type

pandas.core.frame.DataFrame

update(*dtrain*, *iteration*, *fobj=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.update)

Update for one iteration, with objective function calculated internally. This function should not be called directly by users.

Parameters

* **dtrain** ([*DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)) – Training data.
* **iteration** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Current iteration number.
* **fobj** (*function*) – Customized objective function.

Return type

None

**Learning API**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.training)

Training Library containing training routines.

xgboost.train(*params*, *dtrain*, *num\_boost\_round=10*, *\**, *evals=None*, *obj=None*, *feval=None*, *maximize=None*, *early\_stopping\_rounds=None*, *evals\_result=None*, *verbose\_eval=True*, *xgb\_model=None*, *callbacks=None*, *custom\_metric=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.train)

Train a booster with given parameters.

Parameters

* **params** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Booster params.
* **dtrain** ([*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)) – Data to be trained.
* **num\_boost\_round** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of boosting iterations.
* **evals** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*,* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]*) – List of validation sets for which metrics will evaluated during training. Validation metrics will help us track the performance of the model.
* **obj** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]]]*) – Custom objective function. See [Custom Objective](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for details.
* **feval** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]*) –

Deprecated since version 1.6.0: Use *custom\_metric* instead.

* **maximize** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to maximize feval.
* **early\_stopping\_rounds** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **evals**. The method returns the model from the last iteration (not the best one). Use custom callback or model slicing if the best model is desired. If there’s more than one item in **evals**, the last entry will be used for early stopping. If there’s more than one metric in the **eval\_metric** parameter given in **params**, the last metric will be used for early stopping. If early stopping occurs, the model will have two additional fields: bst.best\_score, bst.best\_iteration.
* **evals\_result** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

This dictionary stores the evaluation results of all the items in watchlist.

Example: with a watchlist containing [(dtest,'eval'), (dtrain,'train')] and a parameter containing ('eval\_metric': 'logloss'), the **evals\_result** returns

{'train': {'logloss': ['0.48253', '0.35953']},

'eval': {'logloss': ['0.480385', '0.357756']}}

* **verbose\_eval** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – Requires at least one item in **evals**. If **verbose\_eval** is True then the evaluation metric on the validation set is printed at each boosting stage. If **verbose\_eval** is an integer then the evaluation metric on the validation set is printed at every given **verbose\_eval** boosting stage. The last boosting stage / the boosting stage found by using **early\_stopping\_rounds** is also printed. Example: with verbose\_eval=4 and at least one item in **evals**, an evaluation metric is printed every 4 boosting stages, instead of every boosting stage.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*,* [*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*]]*) – Xgb model to be loaded before training (allows training continuation).
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **custom\_metric** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]*) –

Custom metric function. See [Custom Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for details.

Returns

**Booster**

Return type

a trained booster model

xgboost.cv(*params*, *dtrain*, *num\_boost\_round=10*, *nfold=3*, *stratified=False*, *folds=None*, *metrics=()*, *obj=None*, *feval=None*, *maximize=None*, *early\_stopping\_rounds=None*, *fpreproc=None*, *as\_pandas=True*, *verbose\_eval=None*, *show\_stdv=True*, *seed=0*, *callbacks=None*, *shuffle=True*, *custom\_metric=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.cv)

Cross-validation with given parameters.

Parameters

* **params** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)) – Booster params.
* **dtrain** ([*DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)) – Data to be trained.
* **num\_boost\_round** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of boosting iterations.
* **nfold** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of folds in CV.
* **stratified** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Perform stratified sampling.
* **folds** (*a KFold or StratifiedKFold instance or list of fold indices*) – Sklearn KFolds or StratifiedKFolds object. Alternatively may explicitly pass sample indices for each fold. For n folds, **folds** should be a length n list of tuples. Each tuple is (in,out) where in is a list of indices to be used as the training samples for the n th fold and out is a list of indices to be used as the testing samples for the n th fold.
* **metrics** (*string or list of strings*) – Evaluation metrics to be watched in CV.
* **obj** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]]]*) – Custom objective function. See [Custom Objective](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for details.
* **feval** (*function*) –

Deprecated since version 1.6.0: Use *custom\_metric* instead.

* **maximize** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to maximize feval.
* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Activates early stopping. Cross-Validation metric (average of validation metric computed over CV folds) needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. The last entry in the evaluation history will represent the best iteration. If there’s more than one metric in the **eval\_metric** parameter given in **params**, the last metric will be used for early stopping.
* **fpreproc** (*function*) – Preprocessing function that takes (dtrain, dtest, param) and returns transformed versions of those.
* **as\_pandas** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)*, default True*) – Return pd.DataFrame when pandas is installed. If False or pandas is not installed, return np.ndarray
* **verbose\_eval** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*, or None, default None*) – Whether to display the progress. If None, progress will be displayed when np.ndarray is returned. If True, progress will be displayed at boosting stage. If an integer is given, progress will be displayed at every given *verbose\_eval* boosting stage.
* **show\_stdv** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)*, default True*) – Whether to display the standard deviation in progress. Results are not affected, and always contains std.
* **seed** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Seed used to generate the folds (passed to numpy.random.seed).
* **callbacks** –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **shuffle** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Shuffle data before creating folds.
* **custom\_metric** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]*) –

Custom metric function. See [Custom Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for details.

Returns

**evaluation history**

Return type

[list](https://docs.python.org/3.6/library/stdtypes.html#list)(string)

**Scikit-Learn API**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.sklearn)

Scikit-Learn Wrapper interface for XGBoost.

*class* xgboost.XGBRegressor(*\**, *objective='reg:squarederror'*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor)

Bases: xgboost.sklearn.XGBModel, [sklearn.base.RegressorMixin](https://scikit-learn.org/stable/modules/generated/sklearn.base.RegressorMixin.html#sklearn.base.RegressorMixin)

Implementation of the scikit-learn API for XGBoost regression.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Note

Custom objective function

A custom objective function can be provided for the objective parameter. In this case, it should have the signature objective(y\_true, y\_pred) -> grad, hess:

y\_true: array\_like of shape [n\_samples]

The target values

y\_pred: array\_like of shape [n\_samples]

The predicted values

grad: array\_like of shape [n\_samples]

The value of the gradient for each sample point.

hess: array\_like of shape [n\_samples]

The value of the second derivative for each sample point

Return type

None

apply(*X*, *ntree\_limit=0*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.predict).
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.best_score)

The best score obtained by early stopping.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=True*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit)

Fit gradient boosting model.

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Feature matrix
* **y** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Labels
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – instance weights
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, or callable, optional*) –

Deprecated since version 1.6.0: Use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: Use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.set_params) instead.

* **verbose** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, xgboost.sklearn.XGBModel,* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is an array like object storing instance weights for the i-th validation set.
* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.set_params) instead.

Return type

xgboost.sklearn.XGBModel

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

score(*X*, *y*, *sample\_weight=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.score)

Return the coefficient of determination of the prediction.

The coefficient of determination

is defined as , where is the residual sum of squares ((y\_true - y\_pred)\*\* 2).sum() and is the total sum of squares ((y\_true - y\_true.mean()) \*\* 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of *y*, disregarding the input features, would get a

score of 0.0.

Parameters

* **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (n\_samples, n\_samples\_fitted), where n\_samples\_fitted is the number of samples used in the fitting for the estimator.
* **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True values for *X*.
* **sample\_weight** (*array-like of shape (n\_samples,), default=None*) – Sample weights.

Returns

**score** –

of self.predict(X) wrt. *y*.

Return type

[float](https://docs.python.org/3.6/library/functions.html#float)

Notes

The

score used when calling score on a regressor uses multioutput='uniform\_average' from version 0.23 to keep consistent with default value of [r2\_score()](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score). This influences the score method of all the multioutput regressors (except for [MultiOutputRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html#sklearn.multioutput.MultiOutputRegressor)).

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

*class* xgboost.XGBClassifier(*\**, *objective='binary:logistic'*, *use\_label\_encoder=False*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier)

Bases: xgboost.sklearn.XGBModel, [sklearn.base.ClassifierMixin](https://scikit-learn.org/stable/modules/generated/sklearn.base.ClassifierMixin.html#sklearn.base.ClassifierMixin)

Implementation of the scikit-learn API for XGBoost classification.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of boosting rounds.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Note

Custom objective function

A custom objective function can be provided for the objective parameter. In this case, it should have the signature objective(y\_true, y\_pred) -> grad, hess:

y\_true: array\_like of shape [n\_samples]

The target values

y\_pred: array\_like of shape [n\_samples]

The predicted values

grad: array\_like of shape [n\_samples]

The value of the gradient for each sample point.

hess: array\_like of shape [n\_samples]

The value of the second derivative for each sample point

* **use\_label\_encoder** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

None

apply(*X*, *ntree\_limit=0*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.predict).
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.best_score)

The best score obtained by early stopping.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=True*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit)

Fit gradient boosting classifier.

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Feature matrix
* **y** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Labels
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – instance weights
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, or callable, optional*) –

Deprecated since version 1.6.0: Use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: Use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.set_params) instead.

* **verbose** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*,* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is an array like object storing instance weights for the i-th validation set.
* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.set_params) instead.

Return type

[xgboost.sklearn.XGBClassifier](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

predict\_proba(*X*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.predict_proba)

Predict the probability of each *X* example being of a given class.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** (*array\_like*) – Feature matrix.
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** (*array\_like*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying *iteration\_range=(10, 20)*, then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

Returns

a numpy array of shape array-like of shape (n\_samples, n\_classes) with the probability of each data example being of a given class.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

score(*X*, *y*, *sample\_weight=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.score)

Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

* **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples.
* **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True labels for *X*.
* **sample\_weight** (*array-like of shape (n\_samples,), default=None*) – Sample weights.

Returns

**score** – Mean accuracy of self.predict(X) wrt. *y*.

Return type

[float](https://docs.python.org/3.6/library/functions.html#float)

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

*class* xgboost.XGBRanker(*\**, *objective='rank:pairwise'*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker)

Bases: xgboost.sklearn.XGBModel, xgboost.sklearn.XGBRankerMixIn

Implementation of the Scikit-Learn API for XGBoost Ranking.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Note

A custom objective function is currently not supported by XGBRanker. Likewise, a custom metric function is not supported either.

Note

Query group information is required for ranking tasks by either using the *group* parameter or *qid* parameter in *fit* method.

Before fitting the model, your data need to be sorted by query group. When fitting the model, you need to provide an additional array that contains the size of each query group.

For example, if your original data look like:

|  |  |  |
| --- | --- | --- |
| qid | label | features |
| 1 | 0 | x\_1 |
| 1 | 1 | x\_2 |
| 1 | 0 | x\_3 |
| 2 | 0 | x\_4 |
| 2 | 1 | x\_5 |
| 2 | 1 | x\_6 |
| 2 | 1 | x\_7 |

then your group array should be [3, 4]. Sometimes using query id (*qid*) instead of group can be more convenient.

apply(*X*, *ntree\_limit=0*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.predict).
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.best_score)

The best score obtained by early stopping.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *group=None*, *qid=None*, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_group=None*, *eval\_qid=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=False*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit)

Fit gradient boosting ranker

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Feature matrix
* **y** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Labels
* **group** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Size of each query group of training data. Should have as many elements as the query groups in the training data. If this is set to None, then user must provide qid.
* **qid** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Query ID for each training sample. Should have the size of n\_samples. If this is set to None, then user must provide group.
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –

Query group weights

Note

Weights are per-group for ranking tasks

In ranking task, one weight is assigned to each query group/id (not each data point). This is because we only care about the relative ordering of data points within each group, so it doesn’t make sense to assign weights to individual data points.

* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_group** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list in which eval\_group[i] is the list containing the sizes of all query groups in the i-th pair in **eval\_set**.
* **eval\_qid** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list in which eval\_qid[i] is the array containing query ID of i-th pair in **eval\_set**.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, optional*) –

Deprecated since version 1.6.0: use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.set_params) instead.

* **verbose** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*,* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) –

A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is a list of group weights on the i-th validation set.

Note

Weights are per-group for ranking tasks

In ranking task, one weight is assigned to each query group (not each data point). This is because we only care about the relative ordering of data points within each group, so it doesn’t make sense to assign weights to individual data points.

* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.set_params) instead.

Return type

[xgboost.sklearn.XGBRanker](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRanker.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

*class* xgboost.XGBRFRegressor(*\**, *learning\_rate=1.0*, *subsample=0.8*, *colsample\_bynode=0.8*, *reg\_lambda=1e-05*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor)

Bases: [xgboost.sklearn.XGBRegressor](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor)

scikit-learn API for XGBoost random forest regression.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of trees in random forest to fit.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Note

Custom objective function

A custom objective function can be provided for the objective parameter. In this case, it should have the signature objective(y\_true, y\_pred) -> grad, hess:

y\_true: array\_like of shape [n\_samples]

The target values

y\_pred: array\_like of shape [n\_samples]

The predicted values

grad: array\_like of shape [n\_samples]

The value of the gradient for each sample point.

hess: array\_like of shape [n\_samples]

The value of the second derivative for each sample point

Return type

None

apply(*X*, *ntree\_limit=0*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.predict).
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.best_score)

The best score obtained by early stopping.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=True*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit)

Fit gradient boosting model.

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Feature matrix
* **y** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Labels
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – instance weights
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, or callable, optional*) –

Deprecated since version 1.6.0: Use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: Use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.set_params) instead.

* **verbose** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*,* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is an array like object storing instance weights for the i-th validation set.
* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.set_params) instead.

Return type

[xgboost.sklearn.XGBRFRegressor](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

score(*X*, *y*, *sample\_weight=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.score)

Return the coefficient of determination of the prediction.

The coefficient of determination

is defined as , where is the residual sum of squares ((y\_true - y\_pred)\*\* 2).sum() and is the total sum of squares ((y\_true - y\_true.mean()) \*\* 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of *y*, disregarding the input features, would get a

score of 0.0.

Parameters

* **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (n\_samples, n\_samples\_fitted), where n\_samples\_fitted is the number of samples used in the fitting for the estimator.
* **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True values for *X*.
* **sample\_weight** (*array-like of shape (n\_samples,), default=None*) – Sample weights.

Returns

**score** –

of self.predict(X) wrt. *y*.

Return type

[float](https://docs.python.org/3.6/library/functions.html#float)

Notes

The

score used when calling score on a regressor uses multioutput='uniform\_average' from version 0.23 to keep consistent with default value of [r2\_score()](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score). This influences the score method of all the multioutput regressors (except for [MultiOutputRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html#sklearn.multioutput.MultiOutputRegressor)).

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFRegressor.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

*class* xgboost.XGBRFClassifier(*\**, *learning\_rate=1.0*, *subsample=0.8*, *colsample\_bynode=0.8*, *reg\_lambda=1e-05*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier)

Bases: [xgboost.sklearn.XGBClassifier](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier)

scikit-learn API for XGBoost random forest classification.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of trees in random forest to fit.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Note

Custom objective function

A custom objective function can be provided for the objective parameter. In this case, it should have the signature objective(y\_true, y\_pred) -> grad, hess:

y\_true: array\_like of shape [n\_samples]

The target values

y\_pred: array\_like of shape [n\_samples]

The predicted values

grad: array\_like of shape [n\_samples]

The value of the gradient for each sample point.

hess: array\_like of shape [n\_samples]

The value of the second derivative for each sample point

apply(*X*, *ntree\_limit=0*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.predict).
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.best_score)

The best score obtained by early stopping.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=True*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit)

Fit gradient boosting classifier.

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Feature matrix
* **y** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Labels
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – instance weights
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, or callable, optional*) –

Deprecated since version 1.6.0: Use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: Use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.set_params) instead.

* **verbose** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*,* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is an array like object storing instance weights for the i-th validation set.
* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.set_params) instead.

Return type

[xgboost.sklearn.XGBRFClassifier](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

predict\_proba(*X*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.predict_proba)

Predict the probability of each *X* example being of a given class.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** (*array\_like*) – Feature matrix.
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** (*array\_like*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying *iteration\_range=(10, 20)*, then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

Returns

a numpy array of shape array-like of shape (n\_samples, n\_classes) with the probability of each data example being of a given class.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

score(*X*, *y*, *sample\_weight=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.score)

Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

* **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples.
* **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True labels for *X*.
* **sample\_weight** (*array-like of shape (n\_samples,), default=None*) – Sample weights.

Returns

**score** – Mean accuracy of self.predict(X) wrt. *y*.

Return type

[float](https://docs.python.org/3.6/library/functions.html#float)

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRFClassifier.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

**Plotting API**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.plotting)

Plotting Library.

xgboost.plot\_importance(*booster*, *ax=None*, *height=0.2*, *xlim=None*, *ylim=None*, *title='Feature importance'*, *xlabel='F score'*, *ylabel='Features'*, *fmap=''*, *importance\_type='weight'*, *max\_num\_features=None*, *grid=True*, *show\_values=True*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.plot_importance)

Plot importance based on fitted trees.

Parameters

* **booster** ([*Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, XGBModel or* [*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)) – Booster or XGBModel instance, or dict taken by Booster.get\_fscore()
* **ax** (*matplotlib Axes, default None*) – Target axes instance. If None, new figure and axes will be created.
* **grid** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)*, Turn the axes grids on or off. Default is True (On).*) –
* **importance\_type** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, default "weight"*) –

How the importance is calculated: either “weight”, “gain”, or “cover”

* + ”weight” is the number of times a feature appears in a tree
  + ”gain” is the average gain of splits which use the feature
  + ”cover” is the average coverage of splits which use the feature where coverage is defined as the number of samples affected by the split
* **max\_num\_features** ([*int*](https://docs.python.org/3.6/library/functions.html#int)*, default None*) – Maximum number of top features displayed on plot. If None, all features will be displayed.
* **height** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default 0.2*) – Bar height, passed to ax.barh()
* **xlim** ([*tuple*](https://docs.python.org/3.6/library/stdtypes.html#tuple)*, default None*) – Tuple passed to axes.xlim()
* **ylim** ([*tuple*](https://docs.python.org/3.6/library/stdtypes.html#tuple)*, default None*) – Tuple passed to axes.ylim()
* **title** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, default "Feature importance"*) – Axes title. To disable, pass None.
* **xlabel** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, default "F score"*) – X axis title label. To disable, pass None.
* **ylabel** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, default "Features"*) – Y axis title label. To disable, pass None.
* **fmap** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str) *or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike) *(optional)*) – The name of feature map file.
* **show\_values** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)*, default True*) – Show values on plot. To disable, pass False.
* **kwargs** – Other keywords passed to ax.barh()

Returns

**ax**

Return type

matplotlib Axes

xgboost.plot\_tree(*booster*, *fmap=''*, *num\_trees=0*, *rankdir=None*, *ax=None*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.plot_tree)

Plot specified tree.

Parameters

* **booster** ([*Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, XGBModel*) – Booster or XGBModel instance
* **fmap** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str) *(optional)*) – The name of feature map file
* **num\_trees** ([*int*](https://docs.python.org/3.6/library/functions.html#int)*, default 0*) – Specify the ordinal number of target tree
* **rankdir** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, default "TB"*) – Passed to graphiz via graph\_attr
* **ax** (*matplotlib Axes, default None*) – Target axes instance. If None, new figure and axes will be created.
* **kwargs** – Other keywords passed to to\_graphviz

Returns

**ax**

Return type

matplotlib Axes

xgboost.to\_graphviz(*booster*, *fmap=''*, *num\_trees=0*, *rankdir=None*, *yes\_color=None*, *no\_color=None*, *condition\_node\_params=None*, *leaf\_node\_params=None*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.to_graphviz)

Convert specified tree to graphviz instance. IPython can automatically plot the returned graphiz instance. Otherwise, you should call .render() method of the returned graphiz instance.

Parameters

* **booster** ([*Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, XGBModel*) – Booster or XGBModel instance
* **fmap** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str) *(optional)*) – The name of feature map file
* **num\_trees** ([*int*](https://docs.python.org/3.6/library/functions.html#int)*, default 0*) – Specify the ordinal number of target tree
* **rankdir** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, default "UT"*) – Passed to graphiz via graph\_attr
* **yes\_color** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, default '#0000FF'*) – Edge color when meets the node condition.
* **no\_color** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, default '#FF0000'*) – Edge color when doesn’t meet the node condition.
* **condition\_node\_params** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Condition node configuration for for graphviz. Example:

{'shape': 'box',

'style': 'filled,rounded',

'fillcolor': '#78bceb'}

* **leaf\_node\_params** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Leaf node configuration for graphviz. Example:

{'shape': 'box',

'style': 'filled',

'fillcolor': '#e48038'}

* **\*\*kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) – Other keywords passed to graphviz graph\_attr, e.g. graph [ {key} = {value} ]

Returns

**graph**

Return type

graphviz.Source

**Callback API**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.callback)

Callback library containing training routines. See [Callback Functions](https://xgboost.readthedocs.io/en/stable/python/callbacks.html) for a quick introduction.

*class* xgboost.callback.TrainingCallback[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)

Interface for training callback.

New in version 1.3.0.

Return type

None

after\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback.after_iteration)

Run after each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

after\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback.after_training)

Run after training is finished.

before\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback.before_iteration)

Run before each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

before\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback.before_training)

Run before training starts.

*class* xgboost.callback.EvaluationMonitor(*rank=0*, *period=1*, *show\_stdv=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EvaluationMonitor)

Bases: [xgboost.callback.TrainingCallback](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)

Print the evaluation result at each iteration.

New in version 1.3.0.

Parameters

* **metric** – Extra user defined metric.
* **rank** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Which worker should be used for printing the result.
* **period** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – How many epoches between printing.
* **show\_stdv** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Used in cv to show standard deviation. Users should not specify it.

Return type

None

after\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EvaluationMonitor.after_iteration)

Run after each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

after\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EvaluationMonitor.after_training)

Run after training is finished.

before\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EvaluationMonitor.before_iteration)

Run before each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

before\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EvaluationMonitor.before_training)

Run before training starts.

*class* xgboost.callback.EarlyStopping(*rounds*, *metric\_name=None*, *data\_name=None*, *maximize=None*, *save\_best=False*, *min\_delta=0.0*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping)

Bases: [xgboost.callback.TrainingCallback](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)

Callback function for early stopping

New in version 1.3.0.

Parameters

* **rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Early stopping rounds.
* **metric\_name** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Name of metric that is used for early stopping.
* **data\_name** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Name of dataset that is used for early stopping.
* **maximize** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Whether to maximize evaluation metric. None means auto (discouraged).
* **save\_best** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Whether training should return the best model or the last model.
* **min\_delta** ([*float*](https://docs.python.org/3.6/library/functions.html#float)) –

Minimum absolute change in score to be qualified as an improvement.

New in version 1.5.0.

clf = xgboost.XGBClassifier(tree\_method="gpu\_hist")

es = xgboost.callback.EarlyStopping(

rounds=2,

abs\_tol=1e-3,

save\_best=True,

maximize=False,

data\_name="validation\_0",

metric\_name="mlogloss",

)

X, y = load\_digits(return\_X\_y=True)

clf.fit(X, y, eval\_set=[(X, y)], callbacks=[es])

Return type

None

after\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping.after_iteration)

Run after each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

after\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping.after_training)

Run after training is finished.

before\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping.before_iteration)

Run before each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

before\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping.before_training)

Run before training starts.

*class* xgboost.callback.LearningRateScheduler(*learning\_rates*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.LearningRateScheduler)

Bases: [xgboost.callback.TrainingCallback](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)

Callback function for scheduling learning rate.

New in version 1.3.0.

Parameters

**learning\_rates** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]]*) – If it’s a callable object, then it should accept an integer parameter *epoch* and returns the corresponding learning rate. Otherwise it should be a sequence like list or tuple with the same size of boosting rounds.

Return type

None

after\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.LearningRateScheduler.after_iteration)

Run after each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

after\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.LearningRateScheduler.after_training)

Run after training is finished.

before\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.LearningRateScheduler.before_iteration)

Run before each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

before\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.LearningRateScheduler.before_training)

Run before training starts.

*class* xgboost.callback.TrainingCheckPoint(*directory*, *name='model'*, *as\_pickle=False*, *iterations=100*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCheckPoint)

Bases: [xgboost.callback.TrainingCallback](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)

Checkpointing operation.

New in version 1.3.0.

Parameters

* **directory** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Output model directory.
* **name** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – pattern of output model file. Models will be saved as name\_0.json, name\_1.json, name\_2.json ….
* **as\_pickle** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When set to True, all training parameters will be saved in pickle format, instead of saving only the model.
* **iterations** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Interval of checkpointing. Checkpointing is slow so setting a larger number can reduce performance hit.

Return type

None

after\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCheckPoint.after_iteration)

Run after each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

after\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCheckPoint.after_training)

Run after training is finished.

before\_iteration(*model*, *epoch*, *evals\_log*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCheckPoint.before_iteration)

Run before each iteration. Return True when training should stop.

Parameters

* **epoch** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals\_log** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*],* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]]]*) –

Return type

[bool](https://docs.python.org/3.6/library/functions.html#bool)

before\_training(*model*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCheckPoint.before_training)

Run before training starts.

**Dask API**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.dask)

**Dask extensions for distributed training**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#dask-extensions-for-distributed-training)

See [Distributed XGBoost with Dask](https://xgboost.readthedocs.io/en/stable/tutorials/dask.html) for simple tutorial. Also [XGBoost Dask Feature Walkthrough](https://xgboost.readthedocs.io/en/stable/python/dask-examples/index.html) for some examples.

There are two sets of APIs in this module, one is the functional API including train and predict methods. Another is stateful Scikit-Learner wrapper inherited from single-node Scikit-Learn interface.

The implementation is heavily influenced by dask\_xgboost: <https://github.com/dask/dask-xgboost>

**Optional dask configuration**[**ℑ**](https://xgboost.readthedocs.io/en/stable/python/python_api.html#optional-dask-configuration)

* **xgboost.scheduler\_address**: Specify the scheduler address, see [Troubleshooting](https://xgboost.readthedocs.io/en/stable/tutorials/dask.html#tracker-ip).

New in version 1.6.0.

dask.config.set({"xgboost.scheduler\_address": "192.0.0.100"})

# We can also specify the port.

dask.config.set({"xgboost.scheduler\_address": "192.0.0.100:12345"})

*class* xgboost.dask.DaskDMatrix(*client*, *data*, *label=None*, *\**, *weight=None*, *base\_margin=None*, *missing=None*, *silent=False*, *feature\_names=None*, *feature\_types=None*, *group=None*, *qid=None*, *label\_lower\_bound=None*, *label\_upper\_bound=None*, *feature\_weights=None*, *enable\_categorical=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskDMatrix)

Bases: [object](https://docs.python.org/3.6/library/functions.html#object)

DMatrix holding on references to Dask DataFrame or Dask Array. Constructing a *DaskDMatrix* forces all lazy computation to be carried out. Wait for the input data explicitly if you want to see actual computation of constructing *DaskDMatrix*.

See doc for [xgboost.DMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix) constructor for other parameters. DaskDMatrix accepts only dask collection.

Note

DaskDMatrix does not repartition or move data between workers. It’s the caller’s responsibility to balance the data.

New in version 1.0.0.

Parameters

* **client** ([*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)) – Specify the dask client used for training. Use default client returned from dask if it’s set to None.
* **data** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) –
* **label** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)) –
* **silent** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –
* **feature\_names** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) –
* **feature\_types** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) –
* **group** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **qid** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **label\_lower\_bound** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **label\_upper\_bound** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

None

*class* xgboost.dask.DaskDeviceQuantileDMatrix(*client*, *data*, *label=None*, *\**, *weight=None*, *base\_margin=None*, *missing=None*, *silent=False*, *feature\_names=None*, *feature\_types=None*, *max\_bin=256*, *group=None*, *qid=None*, *label\_lower\_bound=None*, *label\_upper\_bound=None*, *feature\_weights=None*, *enable\_categorical=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskDeviceQuantileDMatrix)

Bases: [xgboost.dask.DaskDMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskDMatrix)

Specialized data type for *gpu\_hist* tree method. This class is used to reduce the memory usage by eliminating data copies. Internally the all partitions/chunks of data are merged by weighted GK sketching. So the number of partitions from dask may affect training accuracy as GK generates bounded error for each merge. See doc string for [xgboost.DeviceQuantileDMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DeviceQuantileDMatrix) and [xgboost.DMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix) for other parameters.

New in version 1.2.0.

Parameters

* **max\_bin** (*Number of bins for histogram construction.*) –
* **client** ([*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)) –
* **data** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) –
* **label** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)) –
* **silent** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –
* **feature\_names** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) –
* **feature\_types** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*,* [*List*](https://docs.python.org/3.6/library/typing.html#typing.List)*[*[*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]]]*) –
* **group** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **qid** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **label\_lower\_bound** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **label\_upper\_bound** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

None

xgboost.dask.train(*client*, *params*, *dtrain*, *num\_boost\_round=10*, *\**, *evals=None*, *obj=None*, *feval=None*, *early\_stopping\_rounds=None*, *xgb\_model=None*, *verbose\_eval=True*, *callbacks=None*, *custom\_metric=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.train)

Train XGBoost model.

New in version 1.0.0.

Note

Other parameters are the same as [xgboost.train()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.train) except for *evals\_result*, which is returned as part of function return value instead of argument.

Parameters

* **client** ([*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)) – Specify the dask client used for training. Use default client returned from dask if it’s set to None.
* **params** ([*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*]*) –
* **dtrain** ([*xgboost.dask.DaskDMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskDMatrix)) –
* **num\_boost\_round** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –
* **evals** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*xgboost.dask.DaskDMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskDMatrix)*,* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]*) –
* **obj** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]]]*) –
* **feval** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]*) –
* **early\_stopping\_rounds** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*]*) –
* **verbose\_eval** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) –
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –
* **custom\_metric** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*xgboost.core.DMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*float*](https://docs.python.org/3.6/library/functions.html#float)*]]]*) –

Returns

**results** – A dictionary containing trained booster and evaluation history. *history* field is the same as *eval\_result* from *xgboost.train*.

{'booster': xgboost.Booster,

'history': {'train': {'logloss': ['0.48253', '0.35953']},

'eval': {'logloss': ['0.480385', '0.357756']}}}

Return type

[dict](https://docs.python.org/3.6/library/stdtypes.html#dict)

xgboost.dask.predict(*client*, *model*, *data*, *output\_margin=False*, *missing=nan*, *pred\_leaf=False*, *pred\_contribs=False*, *approx\_contribs=False*, *pred\_interactions=False*, *validate\_features=True*, *iteration\_range=(0, 0)*, *strict\_shape=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.predict)

Run prediction with a trained booster.

Note

Using inplace\_predict might be faster when some features are not needed. See [xgboost.Booster.predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.predict) for details on various parameters. When output has more than 2 dimensions (shap value, leaf with strict\_shape), input should be da.Array or DaskDMatrix.

New in version 1.0.0.

Parameters

* **client** ([*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)) – Specify the dask client used for training. Use default client returned from dask if it’s set to None.
* **model** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*],* [*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*,* [*distributed.Future*](https://distributed.dask.org/en/stable/api.html#distributed.Future)*]*) – The trained model. It can be a distributed.Future so user can pre-scatter it onto all workers.
* **data** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.dask.DaskDMatrix*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskDMatrix)*, da.Array, dd.DataFrame, dd.Series]*) – Input data used for prediction. When input is a dataframe object, prediction output is a series.
* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)) – Used when input data is not DaskDMatrix. Specify the value considered as missing.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –
* **pred\_leaf** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –
* **pred\_contribs** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –
* **approx\_contribs** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –
* **pred\_interactions** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –
* **iteration\_range** ([*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –
* **strict\_shape** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Returns

**prediction** – When input data is dask.array.Array or DaskDMatrix, the return value is an array, when input data is dask.dataframe.DataFrame, return value can be dask.dataframe.Series, dask.dataframe.DataFrame, depending on the output shape.

Return type

dask.array.Array/dask.dataframe.Series

xgboost.dask.inplace\_predict(*client*, *model*, *data*, *iteration\_range=(0, 0)*, *predict\_type='value'*, *missing=nan*, *validate\_features=True*, *base\_margin=None*, *strict\_shape=False*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.inplace_predict)

Inplace prediction. See doc in [xgboost.Booster.inplace\_predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.inplace_predict) for details.

New in version 1.1.0.

Parameters

* **client** ([*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)) – Specify the dask client used for training. Use default client returned from dask if it’s set to None.
* **model** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)*],* [*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*,* [*distributed.Future*](https://distributed.dask.org/en/stable/api.html#distributed.Future)*]*) – See [xgboost.dask.predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.predict) for details.
* **data** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – dask collection.
* **iteration\_range** ([*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – See [xgboost.Booster.predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.predict) for details.
* **predict\_type** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)) – See [xgboost.Booster.inplace\_predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.inplace_predict) for details.
* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)) – Value in the input data which needs to be present as a missing value. If None, defaults to np.nan.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –

See [xgboost.DMatrix](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix) for details.

New in version 1.4.0.

* **strict\_shape** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

See [xgboost.Booster.predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster.predict) for details.

New in version 1.4.0.

* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Returns

When input data is dask.array.Array, the return value is an array, when input data is dask.dataframe.DataFrame, return value can be dask.dataframe.Series, dask.dataframe.DataFrame, depending on the output shape.

Return type

prediction

*class* xgboost.dask.DaskXGBClassifier(*max\_depth=None*, *max\_leaves=None*, *max\_bin=None*, *grow\_policy=None*, *learning\_rate=None*, *n\_estimators=100*, *verbosity=None*, *objective=None*, *booster=None*, *tree\_method=None*, *n\_jobs=None*, *gamma=None*, *min\_child\_weight=None*, *max\_delta\_step=None*, *subsample=None*, *sampling\_method=None*, *colsample\_bytree=None*, *colsample\_bylevel=None*, *colsample\_bynode=None*, *reg\_alpha=None*, *reg\_lambda=None*, *scale\_pos\_weight=None*, *base\_score=None*, *random\_state=None*, *missing=nan*, *num\_parallel\_tree=None*, *monotone\_constraints=None*, *interaction\_constraints=None*, *importance\_type=None*, *gpu\_id=None*, *validate\_parameters=None*, *predictor=None*, *enable\_categorical=False*, *max\_cat\_to\_onehot=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *callbacks=None*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier)

Bases: xgboost.dask.DaskScikitLearnBase, [sklearn.base.ClassifierMixin](https://scikit-learn.org/stable/modules/generated/sklearn.base.ClassifierMixin.html#sklearn.base.ClassifierMixin)

Implementation of the scikit-learn API for XGBoost classification.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Return type

None

apply(*X*, *ntree\_limit=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.predict).
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.best_score)

The best score obtained by early stopping.

*property* client*:* [*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.client)

The dask client used in this model. The *Client* object can not be serialized for transmission, so if task is launched from a worker instead of directly from the client process, this attribute needs to be set at that worker.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=True*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit)

Fit gradient boosting model.

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Feature matrix
* **y** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Labels
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – instance weights
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series],* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, or callable, optional*) –

Deprecated since version 1.6.0: Use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: Use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.set_params) instead.

* **verbose** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is an array like object storing instance weights for the i-th validation set.
* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.set_params) instead.

Return type

[DaskXGBClassifier](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

predict\_proba(*X*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.predict_proba)

Predict the probability of each *X* example being of a given class.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** (*array\_like*) – Feature matrix.
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** (*array\_like*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying *iteration\_range=(10, 20)*, then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

Returns

a numpy array of shape array-like of shape (n\_samples, n\_classes) with the probability of each data example being of a given class.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

score(*X*, *y*, *sample\_weight=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.score)

Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

* **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples.
* **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True labels for *X*.
* **sample\_weight** (*array-like of shape (n\_samples,), default=None*) – Sample weights.

Returns

**score** – Mean accuracy of self.predict(X) wrt. *y*.

Return type

[float](https://docs.python.org/3.6/library/functions.html#float)

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

*class* xgboost.dask.DaskXGBRegressor(*max\_depth=None*, *max\_leaves=None*, *max\_bin=None*, *grow\_policy=None*, *learning\_rate=None*, *n\_estimators=100*, *verbosity=None*, *objective=None*, *booster=None*, *tree\_method=None*, *n\_jobs=None*, *gamma=None*, *min\_child\_weight=None*, *max\_delta\_step=None*, *subsample=None*, *sampling\_method=None*, *colsample\_bytree=None*, *colsample\_bylevel=None*, *colsample\_bynode=None*, *reg\_alpha=None*, *reg\_lambda=None*, *scale\_pos\_weight=None*, *base\_score=None*, *random\_state=None*, *missing=nan*, *num\_parallel\_tree=None*, *monotone\_constraints=None*, *interaction\_constraints=None*, *importance\_type=None*, *gpu\_id=None*, *validate\_parameters=None*, *predictor=None*, *enable\_categorical=False*, *max\_cat\_to\_onehot=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *callbacks=None*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor)

Bases: xgboost.dask.DaskScikitLearnBase, [sklearn.base.RegressorMixin](https://scikit-learn.org/stable/modules/generated/sklearn.base.RegressorMixin.html#sklearn.base.RegressorMixin)

Implementation of the Scikit-Learn API for XGBoost.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Return type

None

apply(*X*, *ntree\_limit=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.predict).
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.best_score)

The best score obtained by early stopping.

*property* client*:* [*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.client)

The dask client used in this model. The *Client* object can not be serialized for transmission, so if task is launched from a worker instead of directly from the client process, this attribute needs to be set at that worker.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=True*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit)

Fit gradient boosting model.

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Feature matrix
* **y** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Labels
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – instance weights
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series],* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, or callable, optional*) –

Deprecated since version 1.6.0: Use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: Use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.set_params) instead.

* **verbose** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is an array like object storing instance weights for the i-th validation set.
* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.set_params) instead.

Return type

[DaskXGBRegressor](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

score(*X*, *y*, *sample\_weight=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.score)

Return the coefficient of determination of the prediction.

The coefficient of determination

is defined as , where is the residual sum of squares ((y\_true - y\_pred)\*\* 2).sum() and is the total sum of squares ((y\_true - y\_true.mean()) \*\* 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of *y*, disregarding the input features, would get a

score of 0.0.

Parameters

* **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (n\_samples, n\_samples\_fitted), where n\_samples\_fitted is the number of samples used in the fitting for the estimator.
* **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True values for *X*.
* **sample\_weight** (*array-like of shape (n\_samples,), default=None*) – Sample weights.

Returns

**score** –

of self.predict(X) wrt. *y*.

Return type

[float](https://docs.python.org/3.6/library/functions.html#float)

Notes

The

score used when calling score on a regressor uses multioutput='uniform\_average' from version 0.23 to keep consistent with default value of [r2\_score()](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score). This influences the score method of all the multioutput regressors (except for [MultiOutputRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html#sklearn.multioutput.MultiOutputRegressor)).

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

*class* xgboost.dask.DaskXGBRanker(*\**, *objective='rank:pairwise'*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker)

Bases: xgboost.dask.DaskScikitLearnBase, xgboost.sklearn.XGBRankerMixIn

Implementation of the Scikit-Learn API for XGBoost Ranking.

New in version 1.4.0.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Note

For dask implementation, group is not supported, use qid instead.

apply(*X*, *ntree\_limit=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.predict).
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.best_score)

The best score obtained by early stopping.

*property* client*:* [*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.client)

The dask client used in this model. The *Client* object can not be serialized for transmission, so if task is launched from a worker instead of directly from the client process, this attribute needs to be set at that worker.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *group=None*, *qid=None*, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_group=None*, *eval\_qid=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=False*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit)

Fit gradient boosting ranker

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Feature matrix
* **y** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Labels
* **group** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Size of each query group of training data. Should have as many elements as the query groups in the training data. If this is set to None, then user must provide qid.
* **qid** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Query ID for each training sample. Should have the size of n\_samples. If this is set to None, then user must provide group.
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) –

Query group weights

Note

Weights are per-group for ranking tasks

In ranking task, one weight is assigned to each query group/id (not each data point). This is because we only care about the relative ordering of data points within each group, so it doesn’t make sense to assign weights to individual data points.

* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series],* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_group** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list in which eval\_group[i] is the list containing the sizes of all query groups in the i-th pair in **eval\_set**.
* **eval\_qid** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list in which eval\_qid[i] is the array containing query ID of i-th pair in **eval\_set**.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, optional*) –

Deprecated since version 1.6.0: use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.set_params) instead.

* **verbose** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) –

A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is a list of group weights on the i-th validation set.

Note

Weights are per-group for ranking tasks

In ranking task, one weight is assigned to each query group (not each data point). This is because we only care about the relative ordering of data points within each group, so it doesn’t make sense to assign weights to individual data points.

* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.set_params) instead.

Return type

[DaskXGBRanker](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRanker.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

*class* xgboost.dask.DaskXGBRFRegressor(*\**, *learning\_rate=1*, *subsample=0.8*, *colsample\_bynode=0.8*, *reg\_lambda=1e-05*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor)

Bases: [xgboost.dask.DaskXGBRegressor](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRegressor)

Implementation of the Scikit-Learn API for XGBoost Random Forest Regressor.

New in version 1.4.0.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of trees in random forest to fit.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Note

Custom objective function

A custom objective function can be provided for the objective parameter. In this case, it should have the signature objective(y\_true, y\_pred) -> grad, hess:

y\_true: array\_like of shape [n\_samples]

The target values

y\_pred: array\_like of shape [n\_samples]

The predicted values

grad: array\_like of shape [n\_samples]

The value of the gradient for each sample point.

hess: array\_like of shape [n\_samples]

The value of the second derivative for each sample point

Return type

None

apply(*X*, *ntree\_limit=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.predict).
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.best_score)

The best score obtained by early stopping.

*property* client*:* [*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.client)

The dask client used in this model. The *Client* object can not be serialized for transmission, so if task is launched from a worker instead of directly from the client process, this attribute needs to be set at that worker.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=True*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit)

Fit gradient boosting model.

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Feature matrix
* **y** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Labels
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – instance weights
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series],* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, or callable, optional*) –

Deprecated since version 1.6.0: Use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: Use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.set_params) instead.

* **verbose** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is an array like object storing instance weights for the i-th validation set.
* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.set_params) instead.

Return type

[DaskXGBRFRegressor](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

score(*X*, *y*, *sample\_weight=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.score)

Return the coefficient of determination of the prediction.

The coefficient of determination

is defined as , where is the residual sum of squares ((y\_true - y\_pred)\*\* 2).sum() and is the total sum of squares ((y\_true - y\_true.mean()) \*\* 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of *y*, disregarding the input features, would get a

score of 0.0.

Parameters

* **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (n\_samples, n\_samples\_fitted), where n\_samples\_fitted is the number of samples used in the fitting for the estimator.
* **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True values for *X*.
* **sample\_weight** (*array-like of shape (n\_samples,), default=None*) – Sample weights.

Returns

**score** –

of self.predict(X) wrt. *y*.

Return type

[float](https://docs.python.org/3.6/library/functions.html#float)

Notes

The

score used when calling score on a regressor uses multioutput='uniform\_average' from version 0.23 to keep consistent with default value of [r2\_score()](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score). This influences the score method of all the multioutput regressors (except for [MultiOutputRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html#sklearn.multioutput.MultiOutputRegressor)).

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFRegressor.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –

*class* xgboost.dask.DaskXGBRFClassifier(*\**, *learning\_rate=1*, *subsample=0.8*, *colsample\_bynode=0.8*, *reg\_lambda=1e-05*, *\*\*kwargs*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier)

Bases: [xgboost.dask.DaskXGBClassifier](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBClassifier)

Implementation of the Scikit-Learn API for XGBoost Random Forest Classifier.

New in version 1.4.0.

Parameters

* **n\_estimators** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Number of trees in random forest to fit.
* **max\_depth** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Maximum tree depth for base learners.
* **max\_leaves** – Maximum number of leaves; 0 indicates no limit.
* **max\_bin** – If using histogram-based algorithm, maximum number of bins per feature
* **grow\_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
* **learning\_rate** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Boosting learning rate (xgb’s “eta”)
* **verbosity** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
* **objective** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*Callable*](https://docs.python.org/3.6/library/typing.html#typing.Callable)*[[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*],* [*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)*]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
* **booster** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which booster to use: gbtree, gblinear or dart.
* **tree\_method** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It’s recommended to study this option from the parameters document [tree method](https://xgboost.readthedocs.io/en/stable/treemethod.html)
* **n\_jobs** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
* **gamma** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – (min\_split\_loss) Minimum loss reduction required to make a further partition on a leaf node of the tree.
* **min\_child\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Minimum sum of instance weight(hessian) needed in a child.
* **max\_delta\_step** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Maximum delta step we allow each tree’s weight estimation to be.
* **subsample** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of the training instance.
* **sampling\_method** –

Sampling method. Used only by *gpu\_hist* tree method.

* + *uniform*: select random training instances uniformly.
  + *gradient\_based* select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
* **colsample\_bytree** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns when constructing each tree.
* **colsample\_bylevel** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each level.
* **colsample\_bynode** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Subsample ratio of columns for each split.
* **reg\_alpha** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L1 regularization term on weights (xgb’s alpha).
* **reg\_lambda** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – L2 regularization term on weights (xgb’s lambda).
* **scale\_pos\_weight** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – Balancing of positive and negative weights.
* **base\_score** (*Optional[*[*float*](https://docs.python.org/3.6/library/functions.html#float)*]*) – The initial prediction score of all instances, global bias.
* **random\_state** (*Optional[Union[*[*numpy.random.RandomState*](https://numpy.org/doc/stable/reference/random/legacy.html#numpy.random.RandomState)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Random number seed.

Note

Using gblinear booster with shotgun updater is nondeterministic as it uses Hogwild algorithm.

* **missing** ([*float*](https://docs.python.org/3.6/library/functions.html#float)*, default np.nan*) – Value in the data which needs to be present as a missing value.
* **num\_parallel\_tree** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Used for boosting random forest.
* **monotone\_constraints** (*Optional[Union[Dict[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*],* [*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]*) – Constraint of variable monotonicity. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/monotonic.html) for more information.
* **interaction\_constraints** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[Tuple[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]]]]*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nested list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other. See [tutorial](https://xgboost.readthedocs.io/en/stable/tutorials/feature_interaction_constraint.html) for more information
* **importance\_type** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) –

The feature importance type for the feature\_importances\_ property:

* + For tree model, it’s either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.
  + For linear model, only “weight” is defined and it’s the normalized coefficients without bias.
* **gpu\_id** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Device ordinal.
* **validate\_parameters** (*Optional[*[*bool*](https://docs.python.org/3.6/library/functions.html#bool)*]*) – Give warnings for unknown parameter.
* **predictor** (*Optional[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*]*) – Force XGBoost to use specific predictor, available choices are [cpu\_predictor, gpu\_predictor].
* **enable\_categorical** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

New in version 1.5.0.

Note

This parameter is experimental

Experimental support for categorical data. When enabled, cudf/pandas.DataFrame should be used to specify categorical data type. Also, JSON/UBJSON serialization format is required.

* **max\_cat\_to\_onehot** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Note

This parameter is experimental

A threshold for deciding whether XGBoost should use one-hot encoding based split for categorical data. When number of categories is lesser than the threshold then one-hot encoding is chosen, otherwise the categories will be partitioned into children nodes. Only relevant for regression and binary classification. See [Categorical Data](https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html) for details.

* **eval\_metric** (*Optional[Union[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, List[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*], Callable]]*) –

New in version 1.6.0.

Metric used for monitoring the training result and early stopping. It can be a string or list of strings as names of predefined metric in XGBoost (See doc/parameter.rst), one of the metrics in [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), or any other user defined metric that looks like *sklearn.metrics*.

If custom objective is also provided, then custom metric should implement the corresponding reverse link function.

Unlike the *scoring* parameter commonly used in scikit-learn, when a callable object is provided, it’s assumed to be a cost function and by default XGBoost will minimize the result during early stopping.

For advanced usage on Early stopping like directly choosing to maximize instead of minimize, see [xgboost.callback.EarlyStopping](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.EarlyStopping).

See [Custom Objective and Evaluation Metric](https://xgboost.readthedocs.io/en/stable/tutorials/custom_metric_obj.html) for more.

Note

This parameter replaces *eval\_metric* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit) method. The old one receives un-transformed prediction regardless of whether custom objective is being used.

from sklearn.datasets import load\_diabetes

from sklearn.metrics import mean\_absolute\_error

X, y = load\_diabetes(return\_X\_y=True)

reg = xgb.XGBRegressor(

tree\_method="hist",

eval\_metric=mean\_absolute\_error,

)

reg.fit(X, y, eval\_set=[(X, y)])

* **early\_stopping\_rounds** (*Optional[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) –

New in version 1.6.0.

Activates early stopping. Validation metric needs to improve at least once in every **early\_stopping\_rounds** round(s) to continue training. Requires at least one item in **eval\_set** in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit).

The method returns the model from the last iteration (not the best one). If there’s more than one item in **eval\_set**, the last entry will be used for early stopping. If there’s more than one metric in **eval\_metric**, the last metric will be used for early stopping.

If early stopping occurs, the model will have three additional fields: [best\_score](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.best_score), [best\_iteration](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.best_iteration) and best\_ntree\_limit.

Note

This parameter replaces *early\_stopping\_rounds* in [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit) method.

* **callbacks** (*Optional[List[*[*TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using [Callback API](https://xgboost.readthedocs.io/en/stable/python/python_api.html#callback-api).

Note

States in callback are not preserved during training, which means callback objects can not be reused for multiple training sessions without reinitialization or deepcopy.

for params in parameters\_grid:

# be sure to (re)initialize the callbacks before each run

callbacks = [xgb.callback.LearningRateScheduler(custom\_rates)]

xgboost.train(params, Xy, callbacks=callbacks)

* **kwargs** ([*dict*](https://docs.python.org/3.6/library/stdtypes.html#dict)*, optional*) –

Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found [here](https://xgboost.readthedocs.io/en/stable/parameter.html). Attempting to set a parameter via the constructor args and \*\*kwargs dict simultaneously will result in a TypeError.

Note

\*\*kwargs unsupported by scikit-learn

\*\*kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

Note

Custom objective function

A custom objective function can be provided for the objective parameter. In this case, it should have the signature objective(y\_true, y\_pred) -> grad, hess:

y\_true: array\_like of shape [n\_samples]

The target values

y\_pred: array\_like of shape [n\_samples]

The predicted values

grad: array\_like of shape [n\_samples]

The value of the gradient for each sample point.

hess: array\_like of shape [n\_samples]

The value of the second derivative for each sample point

Return type

None

apply(*X*, *ntree\_limit=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.apply)

Return the predicted leaf every tree for each sample. If the model is trained with early stopping, then *best\_iteration* is used automatically.

Parameters

* **X** (*array\_like, shape=[n\_samples, n\_features]*) – Input features matrix.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – See [predict()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.predict).
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use iteration\_range instead.

Returns

**X\_leaves** – For each datapoint x in X and for each tree, return the index of the leaf x ends up in. Leaves are numbered within [0; 2\*\*(self.max\_depth+1)), possibly with gaps in the numbering.

Return type

array\_like, shape=[n\_samples, n\_trees]

*property* best\_iteration*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.best_iteration)

The best iteration obtained by early stopping. This attribute is 0-based, for instance if the best iteration is the first round, then best\_iteration is 0.

*property* best\_score*:* [*float*](https://docs.python.org/3.6/library/functions.html#float)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.best_score)

The best score obtained by early stopping.

*property* client*:* [*distributed.Client*](https://distributed.dask.org/en/stable/api.html#distributed.Client)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.client)

The dask client used in this model. The *Client* object can not be serialized for transmission, so if task is launched from a worker instead of directly from the client process, this attribute needs to be set at that worker.

*property* coef\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.coef_)

Coefficients property

Note

Coefficients are defined only for linear learners

Coefficients are only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**coef\_**

Return type

array of shape [n\_features] or [n\_classes, n\_features]

evals\_result()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.evals_result)

Return the evaluation results.

If **eval\_set** is passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit) function, you can call evals\_result() to get evaluation results for all passed **eval\_sets**. When **eval\_metric** is also passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit) function, the **evals\_result** will contain the **eval\_metrics** passed to the [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit) function.

The returned evaluation result is a dictionary:

{'validation\_0': {'logloss': ['0.604835', '0.531479']},

'validation\_1': {'logloss': ['0.41965', '0.17686']}}

Return type

evals\_result

*property* feature\_importances\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.feature_importances_)

Feature importances property, return depends on *importance\_type* parameter.

Returns

* **feature\_importances\_** (array of shape [n\_features] except for multi-class)
* linear model, which returns an array with shape *(n\_features, n\_classes)*

*property* feature\_names\_in\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.feature_names_in_)

Names of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit). Defined only when *X* has feature names that are all strings.

fit(*X*, *y*, *\**, *sample\_weight=None*, *base\_margin=None*, *eval\_set=None*, *eval\_metric=None*, *early\_stopping\_rounds=None*, *verbose=True*, *xgb\_model=None*, *sample\_weight\_eval\_set=None*, *base\_margin\_eval\_set=None*, *feature\_weights=None*, *callbacks=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit)

Fit gradient boosting model.

Note that calling fit() multiple times will cause the model object to be re-fit from scratch. To resume training from a previous checkpoint, explicitly pass xgb\_model argument.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Feature matrix
* **y** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Labels
* **sample\_weight** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – instance weights
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – global bias for each instance.
* **eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series],* [*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]]*) – A list of (X, y) tuple pairs to use as validation sets, for which metrics will be computed. Validation metrics will help us track the performance of the model.
* **eval\_metric** ([*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*, list of str, or callable, optional*) –

Deprecated since version 1.6.0: Use *eval\_metric* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.set_params) instead.

* **early\_stopping\_rounds** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) –

Deprecated since version 1.6.0: Use *early\_stopping\_rounds* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.set_params) instead.

* **verbose** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
* **xgb\_model** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*xgboost.core.Booster*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.Booster)*, xgboost.sklearn.XGBModel]]*) – file name of stored XGBoost model or ‘Booster’ instance XGBoost model to be loaded before training (allows training continuation).
* **sample\_weight\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [L\_1, L\_2, …, L\_n], where each L\_i is an array like object storing instance weights for the i-th validation set.
* **base\_margin\_eval\_set** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]]*) – A list of the form [M\_1, M\_2, …, M\_n], where each M\_i is an array like object storing base margin for the i-th validation set.
* **feature\_weights** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Weight for each feature, defines the probability of each feature being selected when colsample is being used. All values must be greater than 0, otherwise a *ValueError* is thrown.
* **callbacks** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Sequence*](https://docs.python.org/3.6/library/typing.html#typing.Sequence)*[*[*xgboost.callback.TrainingCallback*](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.callback.TrainingCallback)*]]*) –

Deprecated since version 1.6.0: Use *callbacks* in \_\_init\_\_() or [set\_params()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.set_params) instead.

Return type

[DaskXGBRFClassifier](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier)

get\_booster()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.get_booster)

Get the underlying xgboost Booster of this model.

This will raise an exception when fit was not called

Returns

**booster**

Return type

a xgboost booster of underlying model

get\_num\_boosting\_rounds()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.get_num_boosting_rounds)

Gets the number of xgboost boosting rounds.

Return type

[int](https://docs.python.org/3.6/library/functions.html#int)

get\_params(*deep=True*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.get_params)

Get parameters.

Parameters

**deep** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) –

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

get\_xgb\_params()[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.get_xgb_params)

Get xgboost specific parameters.

Return type

[*Dict*](https://docs.python.org/3.6/library/typing.html#typing.Dict)[[str](https://docs.python.org/3.6/library/stdtypes.html#str), [*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)]

*property* intercept\_*:* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.intercept_)

Intercept (bias) property

Note

Intercept is defined only for linear learners

Intercept (bias) is only defined when the linear model is chosen as base learner (*booster=gblinear*). It is not defined for other base learner types, such as tree learners (*booster=gbtree*).

Returns

**intercept\_**

Return type

array of shape (1,) or [n\_classes]

load\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.load_model)

Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from XGBoost format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be loaded when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.load\_model("model.json")

# or

model.load\_model("model.ubj")

Parameters

**fname** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[*[*str*](https://docs.python.org/3.6/library/stdtypes.html#str)*,* [*bytearray*](https://docs.python.org/3.6/library/stdtypes.html#bytearray)*,* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)*]*) – Input file name or memory buffer(see also save\_raw)

Return type

None

*property* n\_features\_in\_*:* [*int*](https://docs.python.org/3.6/library/functions.html#int)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.n_features_in_)

Number of features seen during [fit()](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.fit).

predict(*X*, *output\_margin=False*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.predict)

Predict with *X*. If the model is trained with early stopping, then *best\_iteration* is used automatically. For tree models, when data is on GPU, like cupy array or cuDF dataframe and *predictor* is not specified, the prediction is run on GPU automatically, otherwise it will run on CPU.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** ([*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]*) – Data to predict with.
* **output\_margin** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – Whether to output the raw untransformed margin value.
* **ntree\_limit** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*]*) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Union*](https://docs.python.org/3.6/library/typing.html#typing.Union)*[da.Array, dd.DataFrame, dd.Series]]*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) –

Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying iteration\_range=(10, 20), then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

New in version 1.4.0.

Return type

prediction

predict\_proba(*X*, *ntree\_limit=None*, *validate\_features=True*, *base\_margin=None*, *iteration\_range=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.predict_proba)

Predict the probability of each *X* example being of a given class.

Note

This function is only thread safe for *gbtree* and *dart*.

Parameters

* **X** (*array\_like*) – Feature matrix.
* **ntree\_limit** ([*int*](https://docs.python.org/3.6/library/functions.html#int)) – Deprecated, use *iteration\_range* instead.
* **validate\_features** ([*bool*](https://docs.python.org/3.6/library/functions.html#bool)) – When this is True, validate that the Booster’s and data’s feature\_names are identical. Otherwise, it is assumed that the feature\_names are the same.
* **base\_margin** (*array\_like*) – Margin added to prediction.
* **iteration\_range** ([*Optional*](https://docs.python.org/3.6/library/typing.html#typing.Optional)*[*[*Tuple*](https://docs.python.org/3.6/library/typing.html#typing.Tuple)*[*[*int*](https://docs.python.org/3.6/library/functions.html#int)*,* [*int*](https://docs.python.org/3.6/library/functions.html#int)*]]*) – Specifies which layer of trees are used in prediction. For example, if a random forest is trained with 100 rounds. Specifying *iteration\_range=(10, 20)*, then only the forests built during [10, 20) (half open set) rounds are used in this prediction.

Returns

a numpy array of shape array-like of shape (n\_samples, n\_classes) with the probability of each data example being of a given class.

Return type

prediction

save\_model(*fname*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.save_model)

Save the model to a file.

The model is saved in an XGBoost internal format which is universal among the various XGBoost interfaces. Auxiliary attributes of the Python Booster object (such as feature\_names) will not be saved when using binary format. To save those attributes, use JSON/UBJ instead. See [Model IO](https://xgboost.readthedocs.io/en/stable/tutorials/saving_model.html) for more info.

model.save\_model("model.json")

# or

model.save\_model("model.ubj")

Parameters

**fname** (*string or* [*os.PathLike*](https://docs.python.org/3.6/library/os.html#os.PathLike)) – Output file name

Return type

None

score(*X*, *y*, *sample\_weight=None*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.score)

Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

* **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples.
* **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True labels for *X*.
* **sample\_weight** (*array-like of shape (n\_samples,), default=None*) – Sample weights.

Returns

**score** – Mean accuracy of self.predict(X) wrt. *y*.

Return type

[float](https://docs.python.org/3.6/library/functions.html#float)

set\_params(*\*\*params*)[ℑ](https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.dask.DaskXGBRFClassifier.set_params)

Set the parameters of this estimator. Modification of the sklearn method to allow unknown kwargs. This allows using the full range of xgboost parameters that are not defined as member variables in sklearn grid search.

Return type

self

Parameters

**params** ([*Any*](https://docs.python.org/3.6/library/typing.html#typing.Any)) –