Python API Reference

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

The document in this page is automatically generated by sphinx. The content do not render at github, you can view it at http://xgboost.readthedocs.org/en/latest/python/python_api.html

Core Data Structure

Core XGBoost Library.

class `xgboost.DMatrix`(*data, label=None, missing=0.0, weight=None, silent=False, feature_names=None, feature_types=None*)

Bases: `object`

Data Matrix used in XGBoost.

DMatrix is a internal data structure that used by XGBoost which is optimized for both memory efficiency and training speed. You can construct DMatrix from numpy.arrays

`feature_names`

Get feature names (column labels).

Returns: **feature_names**

Return type: list or None

`feature_types`

Get feature types (column types).

Returns: **feature_types**

Return type: list or None

`get_base_margin`()

Get the base margin of the DMatrix.

Returns:	**base_margin**
Return type:	float

`get_float_info`(*field*)

Get float property from the DMatrix.

Parameters:	**field** (*str*) – The field name of the information
Returns:	**info** – a numpy array of float information of the data
Return type:	array

`get_label`()

Get the label of the DMatrix.

Returns:	**label**
Return type:	array

`get_uint_info`(*field*)

Get unsigned integer property from the DMatrix.

Parameters:	**field** (*str*) – The field name of the information
Returns:	**info** – a numpy array of float information of the data
Return type:	array

`get_weight`()

Get the weight of the DMatrix.

Returns:	**weight**
Return type:	array

`num_col`()

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

Returns:	**number of columns**
Return type:	int

`num_row`()

Get the number of rows in the DMatrix.

Returns: **number of rows**

Return type: int

`save_binary`(*fname*, *silent=True*)

Save DMatrix to an XGBoost buffer.

Parameters:
- **fname** (*string*) – Name of the output buffer file.
- **silent** (*bool (optional; default: True)*) – If set, the output is suppressed.

`set_base_margin`(*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

`set_float_info`(*field*, *data*)

Set float type property into the DMatrix.

Parameters:
- **field** (*str*) – The field name of the information
- **data** (*numpy array*) – The array ofdata to be set

`set_group`(*group*)

Set group size of DMatrix (used for ranking).

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

`set_label`(*label*)

Set label of dmatrix

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

set_uint_info(*field, data*)

Set uint type property into the DMatrix.

Parameters:
- **field** (*str*) – The field name of the information
- **data** (*numpy array*) – The array ofdata to be set

set_weight(*weight*)

Set weight of each instance.

Parameters: **weight** (*array like*) – Weight for each data point

slice(*rindex*)

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

Parameters: **rindex** (*list*) – List of indices to be selected.

Returns: **res** – A new DMatrix containing only selected indices.

Return type: DMatrix

class **xgboost.Booster**(*params=None, cache=(), model_file=None*)

Bases: `object`

"A Booster of of XGBoost.

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

boost(*dtrain, grad, hess*)

Boost the booster for one iteration, with customized gradient statistics.

Parameters:
- **dtrain** (*DMatrix*) – The training DMatrix.
- **grad** (*list*) – The first order of gradient.
- **hess** (*list*) – The second order of gradient.

copy()

Copy the booster object.

Returns: **booster** – a copied booster model

Return type: *Booster*

`dump_model`(*fout*, *fmap=''*, *with_stats=False*)

Dump model into a text file.

Parameters:

- **foout** (*string*) – Output file name.
- **fmap** (*string, optional*) – Name of the file containing feature map names.
- **with_stats** (*bool (optional)*) – Controls whether the split statistics are output.

`eval`(*data*, *name='eval'*, *iteration=0*)

Evaluate the model on mat.

Parameters:

- **data** (*DMatrix*) – The dmatrix storing the input.
- **name** (*str, optional*) – The name of the dataset.
- **iteration** (*int, optional*) – The current iteration number.

Returns: **result** – Evaluation result string.

Return type: str

`eval_set`(*evals*, *iteration=0*, *feval=None*)

Evaluate a set of data.

Parameters:

- **evals** (*list of tuples (DMatrix, string)*) – List of items to be evaluated.
- **iteration** (*int*) – Current iteration.
- **feval** (*function*) – Custom evaluation function.

Returns: **result** – Evaluation result string.

Return type: str

`get_dump`(*fmap=''*, *with_stats=False*)

Returns the dump the model as a list of strings.

`get_fscore`(*fmap=''*)

Get feature importance of each feature.

Parameters: **fmap** (*str (optional)*) – The name of feature map file `load_model`(*fname*)

Load the model from a file.

Parameters: **fname** (*string or a memory buffer*) – Input file name or memory buffer(see also save_raw)

`predict`(*data*, *output_margin=False*, *ntree_limit=0*, *pred_leaf=False*)

Predict with data.

> **NOTE: This function is not thread safe.**

For each booster object, predict can only be called from one thread. If you want to run prediction using multiple thread, call bst.copy() to make copies of model object and then call predict

Parameters:
- **data** (*DMatrix*) – The dmatrix storing the input.
- **output_margin** (*bool*) – Whether to output the raw untransformed margin value.
- **ntree_limit** (*int*) – Limit number of trees in the prediction; defaults to 0 (use all trees).
- **pred_leaf** (*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.

Returns: **prediction**

Return type: numpy array

`save_model`(*fname*)

Save the model to a file.

Parameters: **fname** (*string*) – Output file name

`save_raw`()

Save the model to a in memory buffer represetation

Returns:

Return type: a in memory buffer represetation of the model `set_param`(*params, value=None*)

Set parameters into the Booster.

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

`update`(*dtrain, iteration, fobj=None*)

Update for one iteration, with objective function calculated internally.

Parameters:
- **dtrain** (*DMatrix*) – Training data.
- **iteration** (*int*) – Current iteration number.
- **fobj** (*function*) – Customized objective function.

Learning API

Training Library containing training routines.

`xgboost.train`(*params, dtrain, num_boost_round=10, evals=(), obj=None, feval=None, maximize=False, early_stopping_rounds=None, evals_result=None, verbose_eval=True, learning_rates=None, xgb_model=None*)

Train a booster with given parameters.

Parameters:
- **params** (*dict*) – Booster params.
- **dtrain** (*DMatrix*) – Data to be trained.
- **num_boost_round** (*int*) – Number of boosting iterations.
- **(evals)** (*watchlist*) – List of items to be evaluated during training, this allows user to watch performance on the validation set.
- **obj** (*function*) – Customized objective function.
- **feval** (*function*) – Customized evaluation function.
- **maximize** (*bool*) – Whether to maximize feval.
- **early_stopping_rounds** (*int*) – Activates early stopping. Validation error needs to decrease at least every <early_stopping_rounds> round(s) to continue training. Requires at least one item in evals. If there's more than one, will use the last. Returns the model from the last iteration (not the best one). If early stopping occurs, the model will have three additional fields: bst.best_score, bst.best_iteration and bst.best_ntree_limit. (Use bst.best_ntree_limit to get the correct value if num_parallel_tree and/or num_class appears in the parameters)

- **evals_result** (*dict*) –
  This dictionary stores the evaluation results of all the items in watchlist. Example: with a watchlist containing [(dtest,'eval'), (dtrain,'train')] and and a paramater containing ('eval_metric', 'logloss') Returns: {'train': {'logloss': ['0.48253', '0.35953']},

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

- **verbose_eval** (*bool or 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.
- **learning_rates** (*list or function*) – List of learning rate for each boosting round or a customized function that calculates eta in terms of current number of round and the total number of boosting round (e.g. yields learning rate decay) - list l: eta = l[boosting round] - function f: eta = f(boosting round, num_boost_round)
- **xgb_model** (*file name of stored xgb model or 'Booster' instance*) – Xgb model to be loaded before training (allows training continuation).

Returns: **booster**

Return type: a trained booster model

`xgboost.cv`(*params, dtrain, num_boost_round=10, nfold=3, metrics=(), obj=None, feval=None, maximize=False, early_stopping_rounds=None, fpreproc=None, as_pandas=True, show_progress=None, show_stdv=True, seed=0*)

Cross-validation with given paramaters.

Parameters:

- **params** (*dict*) – Booster params.
- **dtrain** (*DMatrix*) – Data to be trained.
- **num_boost_round** (*int*) – Number of boosting iterations.
- **nfold** (*int*) – Number of folds in CV.
- **metrics** (*string or list of strings*) – Evaluation metrics to be watched in CV.
- **obj** (*function*) – Custom objective function.
- **feval** (*function*) – Custom evaluation function.
- **maximize** (*bool*) – Whether to maximize feval.

- **early_stopping_rounds** (*int*) – Activates early stopping. CV error needs to decrease at least every <early_stopping_rounds> round(s) to continue. Last entry in evaluation history is the one from best iteration.
- **fpreproc** (*function*) – Preprocessing function that takes (dtrain, dtest, param) and returns transformed versions of those.
- **as_pandas** (*bool, default True*) – Return pd.DataFrame when pandas is installed. If False or pandas is not installed, return np.ndarray
- **show_progress** (*bool, 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 *show_progress* boosting stage.
- **show_stdv** (*bool, default True*) – Whether to display the standard deviation in progress. Results are not affected, and always contains std.
- **seed** (*int*) – Seed used to generate the folds (passed to numpy.random.seed).

Returns: **evaluation history**

Return type: list(string)

Scikit-Learn API

Plotting API