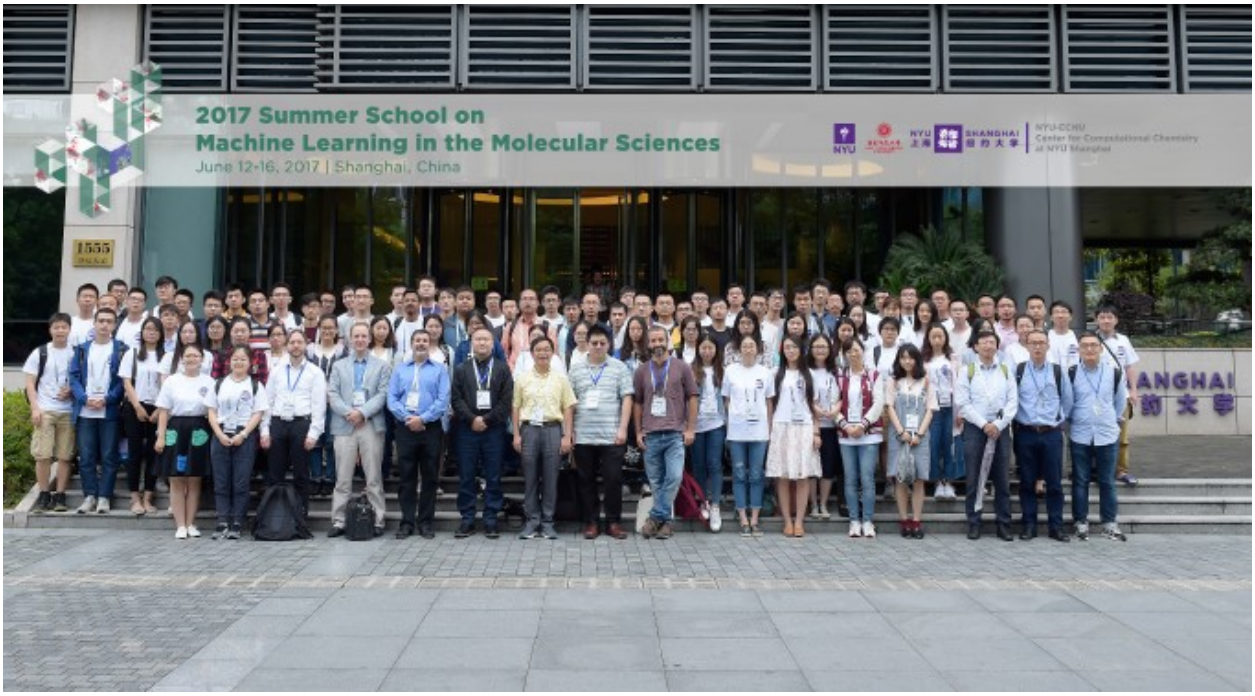
ml-ms

Sep 17, 2020

Introduction to Machine Learning



- **Aims**:

The NYU-ECNU Center for Computational Chemistry at New York University Shanghai (a.k.a, NYU Shanghai) announced a summer school dedicated to machine learning and its applications in the molecular sciences to be held June, 2017 at the NYU Shanghai Pudong Campus. Using a combination of technical lectures and hands-on exercises, the school aimed to instruct attendees in both the fundamentals of modern machine learning techniques and to demonstrate how these approaches can be applied to solve complex computational problems in chemistry, biology, and materials science. In order to promote the idea of free to code, this project is built to help you understand some basic machine learning models mentioned below.

- **Topics**:

Fundamental topics to be covered include basic machine learning models such as *kernel methods* and *neural networks optimization schemes*, *parameter learning* and *delta learning paradigms*, *clustering*, and *decision trees*. Application areas will feature machine learning models for representing and predicting properties of individual molecules and condensed phases, learning algorithms for bypassing explicit quantum chemical and statistical mechanical calculations, and techniques applicable to biomolecular structure prediction, bioinformatics, protein-ligand binding, materials and molecular design and various others.

CHAPTER 1

Introduction to Machine Learning for Molecules and Materials

1.1 Introduction to machine learning

1.2 Learning with kernels

CHAPTER 2

External Links

- genindex
- search

XGBoost: eXtreme Gradient Boosting library.

Contributors: https://github.com/dmlc/xgboost/blob/master/CONTRIBUTORS.md

class xgboost.**DMatrix**(*data*, *label=None*, *weight=None*, *base_margin=None*, *missing=None*, *silent=False*, *feature_names=None*, *feature_types=None*, *nthread=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

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** (*list, numpy 1-D array or cudf.DataFrame, optional*) – Label of the training data.
- **missing** (*float, optional*) – Value in the input data which needs to be present as a missing value. If None, defaults to np.nan.
- **weight** (*list, numpy 1-D array or cudf.DataFrame , optional*) – 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.

- **silent** (*boolean, optional*) – Whether print messages during construction
- **feature_names** (*list, optional*) – Set names for features.
- **feature_types** (*list, optional*) – Set types for features.
- **nthread** (*integer, optional*) – Number of threads to use for loading data when parallelization is applicable. If -1, uses maximum threads available on the system.

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 unsigned integer 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. Saved binary can be later loaded by providing the path to *xgboost.DMatrix()* as input.

Parameters

- **fname** (*string or os.PathLike*) – 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 of data to be set

set_float_info_npy2d(*field*, *data*)

Set float type property into the DMatrix for numpy 2d array input

Parameters

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

set_group(*group*)
Set group size of DMatrix (used for ranking).

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

set_interface_info(*field*, *data*)
Set info type property into DMatrix.

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 of data to be set

set_weight(*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.

---

slice(*rindex*, *allow_groups=False*)
Slice the DMatrix and return a new DMatrix that only contains *rindex*.

Parameters

- **rindex** (*list*) – List of indices to be selected.
- **allow_groups** (*boolean*) – Allow slicing of a matrix with a groups attribute

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

Return type *DMatrix*

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*)
Bases: xgboost.core.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. Implementation does not currently consider weights in quantisation process(unlike DMatrix). Set max_bin to control the number of bins during quantisation.

You can construct DeviceQuantileDMatrix from cupy/cudf/dlpack.

New in version 1.1.0.

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

A Booster of XGBoost.

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

Parameters

- **params** (*dict*) – Parameters for boosters.
- **cache** (*list*) – List of cache items.
- **model_file** (*string or os.PathLike*) – Path to the model file.

attr(*key*)
Get attribute string from the Booster.

Parameters **key** (*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

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*)
Boost the booster for one iteration, with customized gradient statistics. Like `xgboost.core.Booster.update()`, this function should not be called directly by users.

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_format='text'*)
Dump model into a text or JSON file.

Parameters

- **fout** (*string or os.PathLike*) – Output file name.
- **fmap** (*string or os.PathLike, optional*) – Name of the file containing feature map names.
- **with_stats** (*bool, optional*) – Controls whether the split statistics are output.
- **dump_format** (*string, optional*) – Format of model dump file. Can be 'text' or 'json'.

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

feature_names = None

get_dump(*fmap=''*, *with_stats=False*, *dump_format='text'*)
Returns the model dump as a list of strings.

Parameters

- **fmap** (*string or os.PathLike, optional*) – Name of the file containing feature map names.
- **with_stats** (*bool, optional*) – Controls whether the split statistics are output.
- **dump_format** (*string, optional*) – Format of model dump. Can be 'text', 'json' or 'dot'.

get_fscore(*fmap=''*)
Get feature importance of each feature.

Note: Feature importance is defined only for tree boosters

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

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** (*str or os.PathLike (optional)*) – The name of feature map file

get_score(*fmap=''*, *importance_type='weight'*)
Get feature importance of each feature. 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: Feature importance is defined only for tree boosters

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

Parameters

- **fmap** (*str or os.PathLike (optional)*) – The name of feature map file.

- **importance_type** (*str, default 'weight'*) – One of the importance types defined above.

get_split_value_histogram(*feature*, *fmap=''*, *bins=None*, *as_pandas=True*)
Get split value histogram of a feature

Parameters

- **feature** (*str*) – The name of the feature.
- **fmap** (*str or os.PathLike (optional)*) – The name of feature map file.
- **bin** (*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, default True*) – Return pd.DataFrame when pandas is installed. If False or pandas is not installed, return numpy ndarray.

Returns

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

inplace_predict(*data*, *iteration_range=(0, 0)*, *predict_type='value'*, *missing=nan*)
Run prediction in-place, Unlike `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*) – 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) (open set) rounds are used in this prediction.
- **predict_type** (*str*) –
  - *value* Output model prediction values.
  - *margin* Output the raw untransformed margin value.
- **missing** (*float*) – Value in the input data which needs to be present as a missing value.

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*)
Load configuration returned by *save_config*.

New in version 1.0.0.

load_model(*fname*)
Load the model from a file or bytearray. Path to file can be local or as an URI.

The model is loaded from an 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. To preserve all attributes, pickle the Booster object.

Parameters **fname** (*string, os.PathLike, or a memory buffer*) – Input file name or memory buffer(see also save_raw)

load_rabit_checkpoint()
Initialize the model by load from rabit checkpoint.

Returns **version** – The version number of the model.

Return type integer

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*)
Predict with data.

Note:

This function is not thread safe except for gbtree booster.

For gbtree booster, the thread safety is guaranteed by locks. For lock free prediction use inplace_predict instead. Also, the safety does not hold when used in conjunction with other methods.

When using booster other than gbtree, 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.
- **pred_contribs** (*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*) – Approximate the contributions of each feature
- **pred_interactions** (*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*) – 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*) – Whether the prediction value is used for training. This can effect *dart* booster, which performs dropouts during training iterations.

  New in version 1.0.0.

:param .. note:: Using predict() with DART booster: If the booster object is DART type, predict() will not pe
dropouts, i.e. all the trees will be evaluated. If you want to obtain result with dropouts, provide *training=True*.

Returns prediction

Return type numpy array

save_config()
Output internal parameter configuration of Booster as a JSON string.

New in version 1.0.0.

save_model(*fname*)
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. To preserve all attributes, pickle the Booster object.

Parameters fname (*string or os.PathLike*) – Output file name

save_rabit_checkpoint()
Save the current booster to rabit checkpoint.

save_raw()
Save the model to a in memory buffer representation

Returns

Return type a in memory buffer representation of the model

set_attr(***kwargs*)
Set the attribute of the Booster.

Parameters **kwargs – The attributes to set. Setting a value to None deletes an attribute.

set_param(*params*, *value=None*)
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

trees_to_dataframe (*fmap=''*)
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 or os.PathLike (optional)*) – The name of feature map file.

update (*dtrain*, *iteration*, *fobj=None*)
Update for one iteration, with objective function calculated internally. This function should not be called directly by users.

Parameters

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

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*, *xgb_model=None*, *callbacks=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** (*list of pairs (DMatrix, string)*) – List of validation sets for which metrics will evaluated during training. Validation metrics will help us track the performance of the model.
- **obj** (*function*) – Customized objective function.
- **feval** (*function*) – Customized evaluation function.
- **maximize** (*bool*) – Whether to maximize feval.
- **early_stopping_rounds** (*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). 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 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 a parameter containing `('eval_metric': 'logloss')`, the **evals_result** 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.
- **xgb_model** (*file name of stored xgb model or 'Booster' instance*) – Xgb model to be loaded before training (allows training continuation).
- **callbacks** (*list of callback functions*) – List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using Callback API. Example:

```
[xgb.callback.reset_learning_rate(custom_rates)]
```

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=False*, *early_stopping_rounds=None*, *fpreproc=None*, *as_pandas=True*, *verbose_eval=None*, *show_stdv=True*, *seed=0*, *callbacks=None*, *shuffle=True*)

Cross-validation with given parameters.

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.
- **stratified** (*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** (*function*) – Custom objective function.
- **feval** (*function*) – Custom evaluation function.
- **maximize** (*bool*) – Whether to maximize feval.
- **early_stopping_rounds** (*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, default True*) – Return pd.DataFrame when pandas is installed. If False or pandas is not installed, return np.ndarray
- **verbose_eval** (*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 *verbose_eval* 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).
- **callbacks** (*list of callback functions*) – List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using Callback API. Example:

```
[xgb.callback.reset_learning_rate(custom_rates)]
```

- **shuffle** (*bool*) – Shuffle data before creating folds.

Returns evaluation history

Return type list(string)

class xgboost.**RabitTracker**(*hostIP*, *nslave*, *port=9091*, *port_end=9999*)

Bases: object

tracker for rabit

accept_slaves(*nslave*)

alive()

find_share_ring(*tree_map*, *parent_map*, *r*)

get a ring structure that tends to share nodes with the tree return a list starting from r

get_link_map(*nslave*)

get the link map, this is a bit hacky, call for better algorithm to place similar nodes together

static get_neighbor(*rank*, *nslave*)

get_ring(*tree_map*, *parent_map*)

get a ring connection used to recover local data

get_tree(*nslave*)

join()

slave_envs()

get enviroment variables for slaves can be passed in as args or envs

start(*nslave*)

class xgboost.**XGBModel**(*max_depth=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*, *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='gain'*, *gpu_id=None*, *validate_parameters=None*, ***kwargs*)

Bases: sklearn.base.BaseEstimator

Implementation of the Scikit-Learn API for XGBoost.

Parameters

- **n_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
- **max_depth** (*int*) – Maximum tree depth for base learners.
- **learning_rate** (*float*) – Boosting learning rate (xgb's "eta")
- **verbosity** (*int*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
- **objective** (*string or callable*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
- **booster** (*string*) – Specify which booster to use: gbtree, gblinear or dart.
- **tree_method** (*string*) – 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 parameters document.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost.
- **gamma** (*float*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*int*) – Minimum sum of instance weight(hessian) needed in a child.
- **max_delta_step** (*int*) – Maximum delta step we allow each tree's weight estimation to be.
- **subsample** (*float*) – Subsample ratio of the training instance.
- **colsample_bytree** (*float*) – Subsample ratio of columns when constructing each tree.
- **colsample_bylevel** (*float*) – Subsample ratio of columns for each level.
- **colsample_bynode** (*float*) – Subsample ratio of columns for each split.
- **reg_alpha** (*float (xgb's alpha)*) – L1 regularization term on weights
- **reg_lambda** (*float (xgb's lambda)*) – L2 regularization term on weights
- **scale_pos_weight** (*float*) – Balancing of positive and negative weights.
- **base_score** – The initial prediction score of all instances, global bias.
- **random_state** (*int*) – Random number seed.

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

- **missing** (*float, default np.nan*) – Value in the data which needs to be present as a missing value.
- **num_parallel_tree** (*int*) – Used for boosting random forest.
- **monotone_constraints** (*str*) – Constraint of variable monotonicity. See tutorial for more information.

- **interaction_constraints** (*str*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nest 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 for more information
- **importance_type** (*string, default "gain"*) – The feature importance type for the feature_importances_ property: either "gain", "weight", "cover", "total_gain" or "total_cover".
- ****kwargs** (*dict, optional*) – Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found here: https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst. 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*)

Return the predicted leaf every tree for each sample.

Parameters

- **X** (*array_like, shape=[n_samples, n_features]*) – Input features matrix.
- **ntree_limit** (*int*) – Limit number of trees in the prediction; defaults to 0 (use all trees).

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]

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()
Return the evaluation results.

If **eval_set** is passed to the *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* function, the **evals_result** will contain the **eval_metrics** passed to the *fit* function.

Returns evals_result

Return type dictionary

Example

```
param_dist = {'objective':'binary:logistic', 'n_estimators':2}

clf = xgb.XGBModel(**param_dist)

clf.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        eval_metric='logloss',
        verbose=True)

evals_result = clf.evals_result()
```

The variable **evals_result** will contain:

```
{'validation_0': {'logloss': ['0.604835', '0.531479']},
'validation_1': {'logloss': ['0.41965', '0.17686']}}
```

feature_importances_
Feature importances property

Note: Feature importance is defined only for tree boosters

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

Returns feature_importances_

Return type array of shape `[n_features]`

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*, *callbacks=None*)
Fit gradient boosting model

Parameters

- **X** (*array_like*) – Feature matrix
- **y** (*array_like*) – Labels
- **sample_weight** (*array_like*) – instance weights

- **base_margin** (*array_like*) – global bias for each instance.
- **eval_set** (*list, optional*) – 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.
- **sample_weight_eval_set** (*list, optional*) – A list of the form [L_1, L_2, . . . , L_n], where each L_i is a list of instance weights on the i-th validation set.
- **eval_metric** (*str, list of str, or callable, optional*) – If a str, should be a built-in evaluation metric to use. See doc/parameter.rst. If a list of str, should be the list of multiple built-in evaluation metrics to use. If callable, a custom evaluation metric. The call signature is `func(y_predicted, y_true)` where `y_true` will be a DMatrix object such that you may need to call the `get_label` method. It must return a str, value pair where the str is a name for the evaluation and value is the value of the evaluation function. The callable custom objective is always minimized.
- **early_stopping_rounds** (*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 **eval_set**. 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: `clf.best_score`, `clf.best_iteration` and `clf.best_ntree_limit`.
- **verbose** (*bool*) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
- **xgb_model** (*str*) – file name of stored XGBoost model or 'Booster' instance XGBoost model to be loaded before training (allows training continuation).
- **callbacks** (*list of callback functions*) – List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using callback_api. Example:

```
[xgb.callback.reset_learning_rate(custom_rates)]
```

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()
Gets the number of xgboost boosting rounds.

get_params(*deep=True*)
Get parameters.

get_xgb_params()
Get xgboost specific parameters.

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*)
Load the model from a file.

The model is loaded from 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 loaded.

Parameters **fname** (*string*) – Input file name.

predict (*data*, *output_margin=False*, *ntree_limit=None*, *validate_features=True*, *base_margin=None*)
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 `xgb.copy()` to make copies of model object and then call `predict()`.

```
preds = bst.predict(dtest, ntree_limit=num_round)
```

Parameters

- **data** (*numpy.array/scipy.sparse*) – Data to predict with
- **output_margin** (*bool*) – Whether to output the raw untransformed margin value.
- **ntree_limit** (*int*) – Limit number of trees in the prediction; defaults to best_ntree_limit if defined (i.e. it has been trained with early stopping), otherwise 0 (use all trees).
- **validate_features** (*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.

Returns **prediction**

Return type numpy array

save_model (*fname: str*)
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.

Note: See:

https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

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

set_params(***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.

Returns

Return type self

class xgboost.**XGBClassifier**(*objective='binary:logistic'*, ***kwargs*)
Bases: xgboost.sklearn.XGBModel, sklearn.base.ClassifierMixin

Implementation of the scikit-learn API for XGBoost classification.

Parameters

- **max_depth** (*int*) – Maximum tree depth for base learners.
- **learning_rate** (*float*) – Boosting learning rate (xgb's "eta")
- **verbosity** (*int*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
- **objective** (*string or callable*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
- **booster** (*string*) – Specify which booster to use: gbtree, gblinear or dart.
- **tree_method** (*string*) – 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 parameters document.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost.
- **gamma** (*float*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*int*) – Minimum sum of instance weight(hessian) needed in a child.
- **max_delta_step** (*int*) – Maximum delta step we allow each tree's weight estimation to be.
- **subsample** (*float*) – Subsample ratio of the training instance.
- **colsample_bytree** (*float*) – Subsample ratio of columns when constructing each tree.
- **colsample_bylevel** (*float*) – Subsample ratio of columns for each level.
- **colsample_bynode** (*float*) – Subsample ratio of columns for each split.
- **reg_alpha** (*float (xgb's alpha)*) – L1 regularization term on weights
- **reg_lambda** (*float (xgb's lambda)*) – L2 regularization term on weights
- **scale_pos_weight** (*float*) – Balancing of positive and negative weights.
- **base_score** – The initial prediction score of all instances, global bias.
- **random_state** (*int*) – Random number seed.

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

- **missing** (*float, default np.nan*) – Value in the data which needs to be present as a missing value.
- **num_parallel_tree** (*int*) – Used for boosting random forest.
- **monotone_constraints** (*str*) – Constraint of variable monotonicity. See tutorial for more information.
- **interaction_constraints** (*str*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nest 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 for more information
- **importance_type** (*string, default "gain"*) – The feature importance type for the feature_importances_ property: either "gain", "weight", "cover", "total_gain" or "total_cover".
- ****kwargs** (*dict, optional*) – Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found here: https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst. 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

---

evals_result()

Return the evaluation results.

If **eval_set** is passed to the *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* function, the **evals_result** will contain the **eval_metrics** passed to the *fit* function.

Returns **evals_result**

Return type dictionary

Example

```
param_dist = {'objective':'binary:logistic', 'n_estimators':2}

clf = xgb.XGBClassifier(**param_dist)
```

(continues on next page)

(continued from previous page)

```
clf.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        eval_metric='logloss',
        verbose=True)

evals_result = clf.evals_result()
```

The variable **evals_result** will contain

```
{'validation_0': {'logloss': ['0.604835', '0.531479']},
'validation_1': {'logloss': ['0.41965', '0.17686']}}
```

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*, *callbacks=None*)
Fit gradient boosting classifier

Parameters

- **X** (*array_like*) – Feature matrix
- **y** (*array_like*) – Labels
- **sample_weight** (*array_like*) – instance weights
- **base_margin** (*array_like*) – global bias for each instance.
- **eval_set** (*list, optional*) – 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.
- **sample_weight_eval_set** (*list, optional*) – A list of the form [L_1, L_2, . . . , L_n], where each L_i is a list of instance weights on the i-th validation set.
- **eval_metric** (*str, list of str, or callable, optional*) – If a str, should be a built-in evaluation metric to use. See doc/parameter.rst. If a list of str, should be the list of multiple built-in evaluation metrics to use. If callable, a custom evaluation metric. The call signature is `func(y_predicted, y_true)` where `y_true` will be a DMatrix object such that you may need to call the `get_label` method. It must return a str, value pair where the str is a name for the evaluation and value is the value of the evaluation function. The callable custom objective is always minimized.
- **early_stopping_rounds** (*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 **eval_set**. 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: `clf.best_score`, `clf.best_iteration` and `clf.best_ntree_limit`.
- **verbose** (*bool*) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
- **xgb_model** (*str*) – file name of stored XGBoost model or 'Booster' instance XGBoost model to be loaded before training (allows training continuation).
- **callbacks** (*list of callback functions*) – List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using callback_api. Example:

```
[xgb.callback.reset_learning_rate(custom_rates)]
```

predict(*data*, *output_margin=False*, *ntree_limit=None*, *validate_features=True*, *base_margin=None*)
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 `xgb.copy()` to make copies of model object and then call `predict()`.

```
preds = bst.predict(dtest, ntree_limit=num_round)
```

Parameters

- **data** (*array_like*) – 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 best_ntree_limit if defined (i.e. it has been trained with early stopping), otherwise 0 (use all trees).
- **validate_features** (*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.

Returns **prediction**

Return type numpy array

predict_proba(*data*, *ntree_limit=None*, *validate_features=True*, *base_margin=None*)
Predict the probability of each *data* example being of a given class.

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 `xgb.copy()` to make copies of model object and then call predict

---

Parameters

- **data** (DMatrix) – The dmatrix storing the input.
- **ntree_limit** (*int*) – Limit number of trees in the prediction; defaults to best_ntree_limit if defined (i.e. it has been trained with early stopping), otherwise 0 (use all trees).
- **validate_features** (*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.

Returns **prediction** – a numpy array with the probability of each data example being of a given class.

Return type numpy array

class xgboost.**XGBRegressor**(*objective='reg:squarederror'*, ***kwargs*)

Bases: xgboost.sklearn.XGBModel, sklearn.base.RegressorMixin

Implementation of the scikit-learn API for XGBoost regression.

Parameters

- **n_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
- **max_depth** (*int*) – Maximum tree depth for base learners.
- **learning_rate** (*float*) – Boosting learning rate (xgb's "eta")
- **verbosity** (*int*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
- **objective** (*string or callable*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
- **booster** (*string*) – Specify which booster to use: gbtree, gblinear or dart.
- **tree_method** (*string*) – 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 parameters document.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost.
- **gamma** (*float*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*int*) – Minimum sum of instance weight(hessian) needed in a child.
- **max_delta_step** (*int*) – Maximum delta step we allow each tree's weight estimation to be.
- **subsample** (*float*) – Subsample ratio of the training instance.
- **colsample_bytree** (*float*) – Subsample ratio of columns when constructing each tree.
- **colsample_bylevel** (*float*) – Subsample ratio of columns for each level.
- **colsample_bynode** (*float*) – Subsample ratio of columns for each split.
- **reg_alpha** (*float (xgb's alpha)*) – L1 regularization term on weights
- **reg_lambda** (*float (xgb's lambda)*) – L2 regularization term on weights
- **scale_pos_weight** (*float*) – Balancing of positive and negative weights.
- **base_score** – The initial prediction score of all instances, global bias.
- **random_state** (*int*) – Random number seed.

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

- **missing** (*float, default np.nan*) – Value in the data which needs to be present as a missing value.
- **num_parallel_tree** (*int*) – Used for boosting random forest.
- **monotone_constraints** (*str*) – Constraint of variable monotonicity. See tutorial for more information.

- **interaction_constraints** (*str*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nest 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 for more information
- **importance_type** (*string, default "gain"*) – The feature importance type for the feature_importances_ property: either "gain", "weight", "cover", "total_gain" or "total_cover".
- ****kwargs** (*dict, optional*) – Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found here: https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst. 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

---

class xgboost.**XGBRanker**(*objective='rank:pairwise'*, ***kwargs*)

Bases: xgboost.sklearn.XGBModel

Implementation of the Scikit-Learn API for XGBoost Ranking.

Parameters

- **n_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
- **max_depth** (*int*) – Maximum tree depth for base learners.
- **learning_rate** (*float*) – Boosting learning rate (xgb's "eta")
- **verbosity** (*int*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
- **objective** (*string or callable*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
- **booster** (*string*) – Specify which booster to use: gbtree, gblinear or dart.
- **tree_method** (*string*) – 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 parameters document.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost.

- **gamma** (*float*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*int*) – Minimum sum of instance weight(hessian) needed in a child.
- **max_delta_step** (*int*) – Maximum delta step we allow each tree's weight estimation to be.
- **subsample** (*float*) – Subsample ratio of the training instance.
- **colsample_bytree** (*float*) – Subsample ratio of columns when constructing each tree.
- **colsample_bylevel** (*float*) – Subsample ratio of columns for each level.
- **colsample_bynode** (*float*) – Subsample ratio of columns for each split.
- **reg_alpha** (*float (xgb's alpha)*) – L1 regularization term on weights
- **reg_lambda** (*float (xgb's lambda)*) – L2 regularization term on weights
- **scale_pos_weight** (*float*) – Balancing of positive and negative weights.
- **base_score** – The initial prediction score of all instances, global bias.
- **random_state** (*int*) – Random number seed.

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

- **missing** (*float, default np.nan*) – Value in the data which needs to be present as a missing value.
- **num_parallel_tree** (*int*) – Used for boosting random forest.
- **monotone_constraints** (*str*) – Constraint of variable monotonicity. See tutorial for more information.
- **interaction_constraints** (*str*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nest 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 for more information
- **importance_type** (*string, default "gain"*) – The feature importance type for the feature_importances_ property: either "gain", "weight", "cover", "total_gain" or "total_cover".
- ****kwargs** (*dict, optional*) – Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found here: https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst. 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.

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]`.

fit(*X*, *y*, *group*, *sample_weight=None*, *base_margin=None*, *eval_set=None*, *sample_weight_eval_set=None*, *eval_group=None*, *eval_metric=None*, *early_stopping_rounds=None*, *verbose=False*, *xgb_model=None*, *callbacks=None*)
Fit gradient boosting ranker

Parameters

- **X** (*array_like*) – Feature matrix
- **y** (*array_like*) – Labels
- **group** (*array_like*) – Size of each query group of training data. Should have as many elements as the query groups in the training data
- **sample_weight** (*array_like*) – Query group weights

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** (*array_like*) – Global bias for each instance.
- **eval_set** (*list, optional*) – 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.
- **sample_weight_eval_set** (*list, optional*) – 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.

- **eval_group** (*list of arrays, optional*) – 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_metric** (*str, list of str, optional*) – If a str, should be a built-in evaluation metric to use. See doc/parameter.rst. If a list of str, should be the list of multiple built-in evaluation metrics to use. The custom evaluation metric is not yet supported for the ranker.
- **early_stopping_rounds** (*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 **eval_set**. 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: `clf.best_score`, `clf.best_iteration` and `clf.best_ntree_limit`.
- **verbose** (*bool*) – If *verbose* and an evaluation set is used, writes the evaluation metric measured on the validation set to stderr.
- **xgb_model** (*str*) – file name of stored XGBoost model or 'Booster' instance XGBoost model to be loaded before training (allows training continuation).
- **callbacks** (*list of callback functions*) – List of callback functions that are applied at end of each iteration. It is possible to use predefined callbacks by using callback_api. Example:

```
[xgb.callback.reset_learning_rate(custom_rates)]
```

predict (*data*, *output_margin=False*, *ntree_limit=0*, *validate_features=True*, *base_margin=None*)

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 `xgb.copy()` to make copies of model object and then call `predict()`.

```
preds = bst.predict(dtest, ntree_limit=num_round)
```

Parameters

- **data** (*numpy.array/scipy.sparse*) – Data to predict with
- **output_margin** (*bool*) – Whether to output the raw untransformed margin value.
- **ntree_limit** (*int*) – Limit number of trees in the prediction; defaults to best_ntree_limit if defined (i.e. it has been trained with early stopping), otherwise 0 (use all trees).
- **validate_features** (*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.

Returns prediction

Return type numpy array

class xgboost.**XGBRFClassifier**(*learning_rate=1*, *subsample=0.8*, *colsample_bynode=0.8*, *reg_lambda=1e-05*, ***kwargs*)

Bases: `xgboost.sklearn.XGBClassifier`

scikit-learn API for XGBoost random forest classification.

Parameters

- **n_estimators** (*int*) – Number of trees in random forest to fit.
- **max_depth** (*int*) – Maximum tree depth for base learners.
- **learning_rate** (*float*) – Boosting learning rate (xgb's "eta")
- **verbosity** (*int*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
- **objective** (*string or callable*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
- **booster** (*string*) – Specify which booster to use: gbtree, gblinear or dart.
- **tree_method** (*string*) – 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 parameters document.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost.
- **gamma** (*float*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*int*) – Minimum sum of instance weight(hessian) needed in a child.
- **max_delta_step** (*int*) – Maximum delta step we allow each tree's weight estimation to be.
- **subsample** (*float*) – Subsample ratio of the training instance.
- **colsample_bytree** (*float*) – Subsample ratio of columns when constructing each tree.
- **colsample_bylevel** (*float*) – Subsample ratio of columns for each level.
- **colsample_bynode** (*float*) – Subsample ratio of columns for each split.
- **reg_alpha** (*float (xgb's alpha)*) – L1 regularization term on weights
- **reg_lambda** (*float (xgb's lambda)*) – L2 regularization term on weights
- **scale_pos_weight** (*float*) – Balancing of positive and negative weights.
- **base_score** – The initial prediction score of all instances, global bias.
- **random_state** (*int*) – Random number seed.

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

- **missing** (*float, default np.nan*) – Value in the data which needs to be present as a missing value.

- **num_parallel_tree** (*int*) – Used for boosting random forest.
- **monotone_constraints** (*str*) – Constraint of variable monotonicity. See tutorial for more information.
- **interaction_constraints** (*str*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nest 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 for more information
- **importance_type** (*string, default "gain"*) – The feature importance type for the feature_importances_ property: either "gain", "weight", "cover", "total_gain" or "total_cover".
- ****kwargs** (*dict, optional*) – Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found here: https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst. 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

---

get_num_boosting_rounds()
Gets the number of xgboost boosting rounds.

get_xgb_params()
Get xgboost specific parameters.

class xgboost.**XGBRFRegressor**(*learning_rate=1, subsample=0.8, colsample_bynode=0.8, reg_lambda=1e-05, **kwargs*)

Bases: `xgboost.sklearn.XGBRegressor`

scikit-learn API for XGBoost random forest regression.

Parameters

- **max_depth** (*int*) – Maximum tree depth for base learners.
- **learning_rate** (*float*) – Boosting learning rate (xgb's "eta")
- **verbosity** (*int*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
- **objective** (*string or callable*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).

- **booster** (*string*) – Specify which booster to use: gbtree, gblinear or dart.
- **tree_method** (*string*) – 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 parameters document.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost.
- **gamma** (*float*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*int*) – Minimum sum of instance weight(hessian) needed in a child.
- **max_delta_step** (*int*) – Maximum delta step we allow each tree's weight estimation to be.
- **subsample** (*float*) – Subsample ratio of the training instance.
- **colsample_bytree** (*float*) – Subsample ratio of columns when constructing each tree.
- **colsample_bylevel** (*float*) – Subsample ratio of columns for each level.
- **colsample_bynode** (*float*) – Subsample ratio of columns for each split.
- **reg_alpha** (*float (xgb's alpha)*) – L1 regularization term on weights
- **reg_lambda** (*float (xgb's lambda)*) – L2 regularization term on weights
- **scale_pos_weight** (*float*) – Balancing of positive and negative weights.
- **base_score** – The initial prediction score of all instances, global bias.
- **random_state** (*int*) – Random number seed.

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

- **missing** (*float, default np.nan*) – Value in the data which needs to be present as a missing value.
- **num_parallel_tree** (*int*) – Used for boosting random forest.
- **monotone_constraints** (*str*) – Constraint of variable monotonicity. See tutorial for more information.
- **interaction_constraints** (*str*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nest 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 for more information
- **importance_type** (*string, default "gain"*) – The feature importance type for the feature_importances_ property: either "gain", "weight", "cover", "total_gain" or "total_cover".
- ****kwargs** (*dict, optional*) – Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found here: https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst. 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

---

get_num_boosting_rounds()
Gets the number of xgboost boosting rounds.

get_xgb_params()
Get xgboost specific parameters.

xgboost.**plot_importance**(*booster*, *ax=None*, *height=0.2*, *xlim=None*, *ylim=None*, *title='Feature importance'*, *xlabel='F score'*, *ylabel='Features'*, *importance_type='weight'*, *max_num_features=None*, *grid=True*, *show_values=True*, ***kwargs*)
Plot importance based on fitted trees.

Parameters

- **booster** (*Booster, XGBModel or 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, Turn the axes grids on or off. Default is True (On)*) –
- **importance_type** (*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, default None*) – Maximum number of top features displayed on plot. If None, all features will be displayed.
- **height** (*float, default 0.2*) – Bar height, passed to ax.barh()
- **xlim** (*tuple, default None*) – Tuple passed to axes.xlim()
- **ylim** (*tuple, default None*) – Tuple passed to axes.ylim()
- **title** (*str, default "Feature importance"*) – Axes title. To disable, pass None.
- **xlabel** (*str, default "F score"*) – X axis title label. To disable, pass None.

- **ylabel** (*str, default "Features"*) – Y axis title label. To disable, pass None.
- **show_values** (*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*)

Plot specified tree.

Parameters

- **booster** (*Booster, XGBModel*) – Booster or XGBModel instance
- **fmap** (*str (optional)*) – The name of feature map file
- **num_trees** (*int, default 0*) – Specify the ordinal number of target tree
- **rankdir** (*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*)

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, XGBModel*) – Booster or XGBModel instance
- **fmap** (*str (optional)*) – The name of feature map file
- **num_trees** (*int, default 0*) – Specify the ordinal number of target tree
- **rankdir** (*str, default "UT"*) – Passed to graphiz via graph_attr
- **yes_color** (*str, default '#0000FF'*) – Edge color when meets the node condition.
- **no_color** (*str, default '#FF0000'*) – Edge color when doesn't meet the node condition.
- **condition_node_params** (*dict, optional*) – Condition node configuration for for graphviz. Example:

```
{'shape': 'box',
 'style': 'filled,rounded',
 'fillcolor': '#78bceb'}
```

- **leaf_node_params** (*dict, optional*) – Leaf node configuration for graphviz. Example:

```
{'shape': 'box',
 'style': 'filled',
 'fillcolor': '#e48038'}
```

- ****kwargs** (*dict, optional*) – Other keywords passed to graphviz graph_attr, e.g. `graph [ {key} = {value} ]`

Returns **graph**

Return type graphviz.Source

```
import sys
from numpy.distutils.core import Extension, setup
```

Python Module Index

X

Index

L

N

P

R

S

T

U

X