```
Help on package xgboost:

NAME
    xgboost - XGBoost: eXtreme Gradient Boosting library.

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

PACKAGE CONTENTS
    compat
    core
    libpath
    libxgboostwrapper
    plotting
    sklearn
    training

CLASSES
    builtins.object
        xgboost.core.Booster
        xgboost.core.DMatrix
    sklearn.base.BaseEstimator(builtins.object)
        xgboost.sklearn.XGBModel
            xgboost.sklearn.XGBClassifier(xgboost.sklearn.XGBModel,
            sklearn.base.ClassifierMixin)
            xgboost.sklearn.XGBRegressor(xgboost.sklearn.XGBModel,
            sklearn.base.RegressorMixin)

    class Booster(builtins.object)
     |  "A Booster of of XGBoost.
     |
     |  Booster is the model of xgboost, that contains low level routines for
     |  training, prediction and evaluation.
     |
     |  Methods defined here:
     |
     |  __copy__(self)
     |
     |  __deepcopy__(self)
     |
     |  __del__(self)
     |
     |  __getstate__(self)
     |
     |  __init__(self, params=None, cache=(), model_file=None)
     |      Initialize the Booster.
     |
     |      Parameters
     |      ----------
     |      params : dict
     |          Parameters for boosters.
     |      cache : list
     |          List of cache items.
     |      model_file : string
     |          Path to the model file.
     |
     |  __setstate__(self, state)
```

```
|
|  boost(self, 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(self)
|      Copy the booster object.
|
|      Returns
|      -------
|      booster: `Booster`
|        a copied booster model
|
|  dump_model(self, 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(self, 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: str
|          Evaluation result string.
|
|  eval_set(self, 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: str
|          Evaluation result string.
|
|  get_dump(self, fmap='', with_stats=False)
|      Returns the dump the model as a list of strings.
|
|  get_fscore(self, fmap='')
|      Get feature importance of each feature.
|
|      Parameters
|      ----------
|      fmap: str (optional)
|         The name of feature map file
|
|  load_model(self, 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(self, 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 : numpy array
 |
 |  save_model(self, fname)
 |      Save the model to a file.
 |
 |      Parameters
 |      ----------
 |      fname : string
 |          Output file name
 |
 |  save_raw(self)
 |      Save the model to a in memory buffer represetation
 |
 |      Returns
 |      -------
 |      a in memory buffer represetation of the model
 |
 |  set_param(self, 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(self, 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.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  feature_names = None

class DMatrix(builtins.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
|
|  Methods defined here:
|
|  __del__(self)
|
|  __init__(self, data, label=None, missing=0.0, weight=None, silent=False,
feature_names=None, feature_types=None)
|      Data matrix used in XGBoost.
|
|      Parameters
|      ----------
|      data : string/numpy array/scipy.sparse/pd.DataFrame
|          Data source of DMatrix.
|          When data is string type, it represents the path libsvm format txt file,
|          or binary file that xgboost can read from.
|      label : list or numpy 1-D array, optional
|          Label of the training data.
|      missing : float, optional
|          Value in the data which needs to be present as a missing value.
|      weight : list or numpy 1-D array , optional
|          Weight for each instance.
|      silent : boolean, optional
|          Whether print messages during construction
|      feature_names : list, optional
|          Set names for features.
|      feature_types : list, optional
|          Set types for features.
|
|  get_base_margin(self)
|      Get the base margin of the DMatrix.
|
|      Returns
|      -------
|      base_margin : float
|
|  get_float_info(self, field)
|      Get float property from the DMatrix.
|
|      Parameters
|      ----------
|      field: str
|          The field name of the information
|
|      Returns
|      -------
|      info : array
|          a numpy array of float information of the data
|
|  get_label(self)
|      Get the label of the DMatrix.
|
|      Returns
```

```
|       -------
|       label : array
|
|   get_uint_info(self, field)
|       Get unsigned integer property from the DMatrix.
|
|       Parameters
|       ----------
|       field: str
|           The field name of the information
|
|       Returns
|       -------
|       info : array
|           a numpy array of float information of the data
|
|   get_weight(self)
|       Get the weight of the DMatrix.
|
|       Returns
|       -------
|       weight : array
|
|   num_col(self)
|       Get the number of columns (features) in the DMatrix.
|
|       Returns
|       -------
|       number of columns : int
|
|   num_row(self)
|       Get the number of rows in the DMatrix.
|
|       Returns
|       -------
|       number of rows : int
|
|   save_binary(self, 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(self, 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(self, 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(self, group)
|      Set group size of DMatrix (used for ranking).
|
|      Parameters
|      ----------
|      group : array like
|          Group size of each group
|
|  set_label(self, label)
|      Set label of dmatrix
|
|      Parameters
|      ----------
|      label: array like
|          The label information to be set into DMatrix
|
|  set_uint_info(self, 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(self, weight)
|      Set weight of each instance.
|
|      Parameters
|      ----------
|      weight : array like
|          Weight for each data point
|
|  slice(self, rindex)
|      Slice the DMatrix and return a new DMatrix that only contains `rindex`.
|
|      Parameters
|      ----------
|      rindex : list
|          List of indices to be selected.
```

```
 |
 |      Returns
 |      -------
 |      res : DMatrix
 |          A new DMatrix containing only selected indices.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  feature_names
 |      Get feature names (column labels).
 |
 |      Returns
 |      -------
 |      feature_names : list or None
 |
 |  feature_types
 |      Get feature types (column types).
 |
 |      Returns
 |      -------
 |      feature_types : list or None

class XGBClassifier(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")
 |  n_estimators : int
 |      Number of boosted trees to fit.
 |  silent : boolean
 |      Whether to print messages while running boosting.
 |  objective : string
 |      Specify the learning task and the corresponding learning objective.
 |
 |  nthread : 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 split, in each level.
|  reg_alpha : float (xgb's alpha)
|      L2 regularization term on weights
|  reg_lambda : float (xgb's lambda)
|      L1 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.
|  seed : int
|      Random number seed.
|  missing : float, optional
|      Value in the data which needs to be present as a missing value. If
|      None, defaults to np.nan.
|
|  Method resolution order:
|      XGBClassifier
|      XGBModel
|      sklearn.base.BaseEstimator
|      sklearn.base.ClassifierMixin
|      builtins.object
|
|  Methods defined here:
|
|  __init__(self, max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
objective='binary:logistic', nthread=-1, gamma=0, min_child_weight=1, max_delta_step=0,
subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, base_score=0.5, seed=0, missing=None)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  evals_result(self)
|      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 : dictionary
|
|      Example
|      -------
|      param_dist = {'objective':'binary:logistic', 'n_estimators':2}
|
|      clf = xgb.XGBClassifier(**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']}}
|
|  fit(self, X, y, sample_weight=None, eval_set=None, eval_metric=None,
early_stopping_rounds=None, verbose=True)
|      Fit gradient boosting classifier
|
|      Parameters
|      ----------
|      X : array_like
|          Feature matrix
|      y : array_like
|          Labels
|      sample_weight : array_like
|          Weight for each instance
|      eval_set : list, optional
|          A list of (X, y) pairs to use as a validation set for
|          early-stopping
|      eval_metric : str, callable, optional
|          If a str, should be a built-in evaluation metric to use. See
|          doc/parameter.md. 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. This objective is always minimized.
|      early_stopping_rounds : int, optional
|          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 two additional fields: bst.best_score and bst.best_iteration.
|      verbose : bool
|          If `verbose` and an evaluation set is used, writes the evaluation
|          metric measured on the validation set to stderr.
|
|  predict(self, data, output_margin=False, ntree_limit=0)
|
|  predict_proba(self, data, output_margin=False, ntree_limit=0)
|
|  ----------------------------------------------------------------------
|  Methods inherited from XGBModel:
|
|  __setstate__(self, state)
|
|  booster(self)
|      Get the underlying xgboost Booster of this model.
|
|      This will raise an exception when fit was not called
|
|      Returns
|      -------
|      booster : a xgboost booster of underlying model
|
```

```
|  get_params(self, deep=False)
|      Get parameter.s
|
|  get_xgb_params(self)
|      Get xgboost type parameters.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __repr__(self)
|      Return repr(self).
|
|  set_params(self, **params)
|      Set the parameters of this estimator.
|
|      The method works on simple estimators as well as on nested objects
|      (such as pipelines). The former have parameters of the form
|      ``<component>__<parameter>`` so that it's possible to update each
|      component of a nested object.
|
|      Returns
|      -------
|      self
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.BaseEstimator:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Returns 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, shape = (n_samples, n_features)
|          Test samples.
|
|      y : array-like, shape = (n_samples) or (n_samples, n_outputs)
|          True labels for X.
|
|      sample_weight : array-like, shape = [n_samples], optional
|          Sample weights.
|
|      Returns
|      -------
|      score : float
```

```
 |          Mean accuracy of self.predict(X) wrt. y.

class XGBModel(sklearn.base.BaseEstimator)
 |  Implementation of the Scikit-Learn API for XGBoost.
 |
 |  Parameters
 |  ----------
 |  max_depth : int
 |      Maximum tree depth for base learners.
 |  learning_rate : float
 |      Boosting learning rate (xgb's "eta")
 |  n_estimators : int
 |      Number of boosted trees to fit.
 |  silent : boolean
 |      Whether to print messages while running boosting.
 |  objective : string
 |      Specify the learning task and the corresponding learning objective.
 |
 |  nthread : 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 split, in each level.
 |  reg_alpha : float (xgb's alpha)
 |      L2 regularization term on weights
 |  reg_lambda : float (xgb's lambda)
 |      L1 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.
 |  seed : int
 |      Random number seed.
 |  missing : float, optional
 |      Value in the data which needs to be present as a missing value. If
 |      None, defaults to np.nan.
 |
 |  Method resolution order:
 |      XGBModel
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
objective='reg:linear', nthread=-1, gamma=0, min_child_weight=1, max_delta_step=0,
```

```
subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, base_score=0.5, seed=0, missing=None)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __setstate__(self, state)
|
|  booster(self)
|      Get the underlying xgboost Booster of this model.
|
|      This will raise an exception when fit was not called
|
|      Returns
|      -------
|      booster : a xgboost booster of underlying model
|
|  evals_result(self)
|      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 : 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']}}
|
|  fit(self, X, y, eval_set=None, eval_metric=None, early_stopping_rounds=None,
verbose=True)
|      Fit the gradient boosting model
|
|      Parameters
|      ----------
|      X : array_like
|          Feature matrix
|      y : array_like
|          Labels
|      eval_set : list, optional
|          A list of (X, y) tuple pairs to use as a validation set for
|          early-stopping
```

```
 |      eval_metric : str, callable, optional
 |          If a str, should be a built-in evaluation metric to use. See
 |          doc/parameter.md. 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. This objective is always minimized.
 |      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 two additional fields: bst.best_score and bst.best_iteration.
 |      verbose : bool
 |          If `verbose` and an evaluation set is used, writes the evaluation
 |          metric measured on the validation set to stderr.
 |
 |  get_params(self, deep=False)
 |      Get parameter.s
 |
 |  get_xgb_params(self)
 |      Get xgboost type parameters.
 |
 |  predict(self, data, output_margin=False, ntree_limit=0)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as pipelines). The former have parameters of the form
 |      ``<component>__<parameter>`` so that it's possible to update each
 |      component of a nested object.
 |
 |      Returns
 |      -------
 |      self
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.BaseEstimator:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class XGBRegressor(XGBModel, sklearn.base.RegressorMixin)
 |  Implementation of the scikit-learn API for XGBoost regression.
 |      Parameters
```

```
|  ----------
|  max_depth : int
|      Maximum tree depth for base learners.
|  learning_rate : float
|      Boosting learning rate (xgb's "eta")
|  n_estimators : int
|      Number of boosted trees to fit.
|  silent : boolean
|      Whether to print messages while running boosting.
|  objective : string
|      Specify the learning task and the corresponding learning objective.
|
|  nthread : 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 split, in each level.
|  reg_alpha : float (xgb's alpha)
|      L2 regularization term on weights
|  reg_lambda : float (xgb's lambda)
|      L1 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.
|  seed : int
|      Random number seed.
|  missing : float, optional
|      Value in the data which needs to be present as a missing value. If
|      None, defaults to np.nan.
|
|  Method resolution order:
|      XGBRegressor
|      XGBModel
|      sklearn.base.BaseEstimator
|      sklearn.base.RegressorMixin
|      builtins.object
|
|  Methods inherited from XGBModel:
|
|  __init__(self, max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
objective='reg:linear', nthread=-1, gamma=0, min_child_weight=1, max_delta_step=0,
subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, base_score=0.5, seed=0, missing=None)
|      Initialize self.  See help(type(self)) for accurate signature.
|
```

```
|  __setstate__(self, state)
|
|  booster(self)
|      Get the underlying xgboost Booster of this model.
|
|      This will raise an exception when fit was not called
|
|      Returns
|      -------
|      booster : a xgboost booster of underlying model
|
|  evals_result(self)
|      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 : 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']}}
|
|  fit(self, X, y, eval_set=None, eval_metric=None, early_stopping_rounds=None,
verbose=True)
|      Fit the gradient boosting model
|
|      Parameters
|      ----------
|      X : array_like
|          Feature matrix
|      y : array_like
|          Labels
|      eval_set : list, optional
|          A list of (X, y) tuple pairs to use as a validation set for
|          early-stopping
|      eval_metric : str, callable, optional
|          If a str, should be a built-in evaluation metric to use. See
|          doc/parameter.md. 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. This objective is always minimized.
|      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 two additional fields: bst.best_score and bst.best_iteration.
|      verbose : bool
|          If `verbose` and an evaluation set is used, writes the evaluation
|          metric measured on the validation set to stderr.
|
|  get_params(self, deep=False)
|      Get parameter.s
|
|  get_xgb_params(self)
|      Get xgboost type parameters.
|
|  predict(self, data, output_margin=False, ntree_limit=0)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __repr__(self)
|      Return repr(self).
|
|  set_params(self, **params)
|      Set the parameters of this estimator.
|
|      The method works on simple estimators as well as on nested objects
|      (such as pipelines). The former have parameters of the form
|      ``<component>__<parameter>`` so that it's possible to update each
|      component of a nested object.
|
|      Returns
|      -------
|      self
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.BaseEstimator:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.RegressorMixin:
|
|  score(self, X, y, sample_weight=None)
|      Returns the coefficient of determination R^2 of the prediction.
|
|      The coefficient R^2 is defined as (1 - u/v), where u is the regression
```

```
|      sum of squares ((y_true - y_pred) ** 2).sum() and v is the residual
|      sum of squares ((y_true - y_true.mean()) ** 2).sum().
|      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 R^2 score of 0.0.
|
|      Parameters
|      ----------
|      X : array-like, shape = (n_samples, n_features)
|          Test samples.
|
|      y : array-like, shape = (n_samples) or (n_samples, n_outputs)
|          True values for X.
|
|      sample_weight : array-like, shape = [n_samples], optional
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          R^2 of self.predict(X) wrt. y.

FUNCTIONS
    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 : 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 or None, default None
            Whether to display the progress. If None, progress will be displayed
            when np.ndarray is returned.
        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 : list(string)

    plot_importance(booster, ax=None, height=0.2, xlim=None, ylim=None, title='Feature
    importance', xlabel='F score', ylabel='Features', grid=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.
        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.
        kwargs :
            Other keywords passed to ax.barh()

        Returns
        -------
        ax : matplotlib Axes

    plot_tree(booster, num_trees=0, rankdir='UT', ax=None, **kwargs)
        Plot specified tree.

        Parameters
        ----------
        booster : Booster, XGBModel
            Booster or XGBModel instance
        num_trees : int, default 0
            Specify the ordinal number of target tree
        rankdir : str, default "UT"
            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 : matplotlib Axes

to_graphviz(booster, num_trees=0, rankdir='UT', yes_color='#0000FF', no_color='#FF0000',
**kwargs)
    Convert specified tree to graphviz instance. IPython can automatically plot the
    returned graphiz instance. Otherwise, you shoud call .render() method
    of the returned graphiz instance.

    Parameters
    ----------
    booster : Booster, XGBModel
        Booster or XGBModel instance
    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 condigion.
    no_color : str, default '#FF0000'
        Edge color when doesn't meet the node condigion.
    kwargs :
        Other keywords passed to graphviz graph_attr

    Returns
    -------
    ax : matplotlib Axes

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.
    watchlist (evals): list of pairs (DMatrix, string)
        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 : a trained booster model

DATA
    __all__ = ['DMatrix', 'Booster', 'train', 'cv', 'XGBModel', 'XGBClassi...
    __warningregistry__ = {'version': 207, ("unclosed file <_io.TextIOWrap...

VERSION
    0.4

FILE

    /home/shuangyangwang/anaconda3/lib/python3.5/site-packages/xgboost-0.4-py3.5.egg/xgboost/_
    _init__.py
```