GBDTMO

Release 0.0.1

Dec 02, 2021

Contents:

GBDTMO is a gradient boosted decision tree method which supports learning multiple outputs within a single tree. **GBDTMO** constructs the predicts of all outputs or a subset of automatically selected outputs on a leaf. Compared with GBDT for single output, **GBDTMO** has better generalization ability and faster training speed. See our paper for technical details.

CHAPTER 1

Python API

1.1 load_lib

load_lib(path)

Parameters

- path (string): path of gbdtmo.so

Return Python warper of gbdtmo.so

1.2 create_graph

create_graph(file_name, tree_index=0, value_list=[]) This function generate a `Digraph` instance of `graphviz`. You can render it by yourself.

Parameters

- file_name (string): path of the dumped tree.
- tree_index (int): the index (start from 0) of tree to be plotted.
- value_list (list): list of index of output variables to be plotted. Only for **GBDTMO**. When set to `[]`, all outputs variables will be considered.

Return a `Digraph` instance of a learned tree.

1.3 GBDTMulti

GBDTMulti(lib, out_dim=1, params={}) Create an instance of GBDTMO model.

__init__(lib, out_dim, params={})

Parameters

- lib: a Python warper of library by `load_lib`.
- out_dim(int): dimension of output.
- params(dict): a set of parameters. If a parameter is not contained here, it is set to its default value.

set_data(train_set=(), eval_set=()) Set training and eval datasets. eval_set can be missing. Histograms will be constructed and predictions will be initialized.

Parameters

- train_set(tuple): a tuple of numpy array (x_data, x_label). x_data must be *double* and 2D array. If you don't set label, x_label should be *None*. Otherwise, x_label must be *double* or *int32*.
- eval_set(tuple, default=None): the same as train_set.

_set_gh(self, g, h) Set gradient and hessian for growth next tree. *Only used for user-defined loss*.

Parameters

- g(numpy.array): gradient
- h(numpy.array): hessian

_set_label(x, is_train) Reset label. Sometimes it avoids the re-construction of histogram.

Parameters

- x(numpy.array): labels.
- is_train(bool): if true, set labels for train_set else for eval_set.

boost() Growth a new tree after running `_set_gh`.

train(num) training the model from scratch.

Parameters

- num(int): number of boost round.

dump(path) dump the model into a text file which has the following structure:

```
Booster[i]:
  decision node M
  ...
  decision node 1
    leaf node 1
    ...
    leaf node N
Booster[i+1]:
  ...
```

For a decision node:

```
node index, parent, left, right, split column, split value
```

For a leaf node:

```
leaf index, w_0, w_1, ..., w_n
```

Parameters

- path(string): **must be binary coding**. For example, b"tree.txt".

load(path) load the model from a text file.

Parameters

- path(string): **must be binary coding**. For example, b"tree.txt".

predict(x, num_trees=0)

Parameters

- x(numpy.array): input features
- num_trees(int): number of trees used to compute the prediction. If 0, all trees will be used.

Return prediction of x.

1.4 GBDTSingle

GBDTSO is our own implementation of GBDT for single output. It is used to compare the training speed and accuracy with GBDTMO.

GBDTSingle(lib, out_dim, params={}) Create an instance of GBDTSO model. Most of method is shared with GBDTSO. Here we only list the specific methods of GBDTSO.

train_multi(num) training the model from scratch.

Parameters

- num(int): number of boost round. In each round, `out_dim` of trees will be constructed. They correspond to output variables in order.

reset() clear the learned trees and re-initialize the predictions to `base_score`.

CHAPTER 2

Parameters

This page contains descriptions of all parameters in GBDTMO.

2.1 Meta

- **`verbose`: default = True, type = bool**
  - If True, print loss information every round. Otherwise, print nothing.
- **`seed`: default = 0, type = int.**
  - Random seed. **No effect currently**.
- **`num_threads`: default = 2, type = int.**
  - Number of threads for training.
- **`hist_cache`: default = 16, type = int.**
  - Maximum number of histogram cache
- **`topk`: default = 0.**
  - Sparse factors for sparse split finding.
  - If 0, non-sparse split finding is used.
- **`one_side`: default = True, type = bool.**
  - Algorithm type for sparse split finding.
  - If True, the restricted one is used.
  - Only used when *topk* not equal to 0.
- **`max_bins`: default = 32, type = int.**
  - Maximum number of bins for each input variable.

2.2 Tree

- **`max_depth`: default = 4, type = int.**
  - Maximum depth of trees, at least 1.
- **`max_leaves`: default = 32, type = int.**
  - Maximum leaves of each tree.
- **`min_samples`: default = 20, type = int.**
  - Minimum number of samples of each leaf.
  - Stop growth if current number of samples smaller than this value.
- **`early_stop`: default = 0, type = int.**
  - Number of rounds for early stop.
  - If 0, early stop is not used.

2.3 Learning

- **`base_score`: default = 0.0, type = double.**
  - Initial value of prediction.
- **`subsample`: default = 1.0, type = double.**
  - Column sample rate. **No effect currently**.
- **`lr`: default = 0.2, type = double.**
  - Learning rate.
- **`reg_l1`: default = 0.0, type = double.**
  - L1 regularization.
  - Not used for sparse split finding currently.
- **`reg_l2`: default = 1.0, type = double.**
  - L2 regularization.
- **`gamma`: default = 1e-3, type = double.**
  - Minimum objective gain to split.
- **`loss`: default = 'mse', type = string.**
  - **Must be binary coding**. For example, b'mse' in Python.
  - Must be one of 'mse' (mean square error), 'bce' (binary cross entropy), 'ce' (cross entropy), and 'ce_column' (only for `GBDTSingle`).

CHAPTER 3

Examples

3.1 Plotting tree

Suppose the model is dumped into `gbdtmo.txt`, plot 5th tree by:

```
>>> from gbdtmo import create_graph
>>> from graphviz import Digraph
>>> graph = create_graph("gbdtmo.txt", 5, [0, 3])
>>> graph.render("tree_5", format='pdf')
```

Then `tree_5.pdf` will be generated.

3.2 Using GBDTMO

First import `gbdtmo`

```
>>> from gbdtmo import GBDTMulti, load_lib
```

Load from `gbdtmo.so`

```
>>> LIB = load_lib("path to gbdtmo.so")
```

Build an instance of GBDTMO. Here the `out_dim` is set to 10 and MSE loss is used.

```
>>> inp_dim, out_dim = 10, 5
>>> params = {"max_depth": 5, "lr": 0.1, 'loss': b"mse"}
>>> booster = GBDTMulti(LIB, out_dim=out_dim, params=params)
```

Set the training and eval datasets.

```
>>> x_train, y_train = np.random.rand(10000, inp_dim), np.random.rand(10000, out_dim)
>>> x_valid, y_valid = np.random.rand(10000, inp_dim), np.random.rand(10000, out_dim)
>>> booster.set_data((x_train, y_train), (x_valid, y_valid))
```

Training with 30 rounds and dump it into text file.

```
>>> booster.train(30)
>>> booster.dump(b"tree.txt")
```

3.3 Custom loss

We show how to train GBDTMO via custom loss. Here is an example of MSE.

```
def MSE(x, y):
  g = x - y
  h = np.ones_like(x)
  return g, h
```

```
>>> g, h = MSE(booster.preds_train.copy(), booster.label.copy())
>>> booster._set_gh(g, h)
>>> booster.boost()
```

In this way, a new tree is constructed and the predictions are updated.

Feel free to contact with us if you have any questions or suggestions.