

# XGBoost Basics

Learn about the basics of using XGBoost.

## Chapter Goals:

- Learn about the XGBoost data matrix
- Train a `Booster` object in XGBoost

### A. Basic data structures

The basic data structure for XGBoost is the `DMatrix`

([https://xgboost.readthedocs.io/en/latest/python/python\\_api.html#xgboost.DMatrix](https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.DMatrix)), which represents a data matrix. The `DMatrix` can be constructed from NumPy arrays.

The code below creates `DMatrix` objects with and without labels.

```
1 data = np.array([
2     [1.2, 3.3, 1.4],
3     [5.1, 2.2, 6.6]])
4
5 import xgboost as xgb
6 dmat1 = xgb.DMatrix(data)
7
8 labels = np.array([0, 1])
9 dmat2 = xgb.DMatrix(data, label=labels)
```

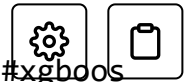


✓ Succeeded

The `DMatrix` object can be used for training and using a `Booster`

([https://xgboost.readthedocs.io/en/latest/python/python\\_api.html#xgboost.Booster](https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.Booster)) object, which represents the gradient boosted decision tree.

The train



([https://xgboost.readthedocs.io/en/latest/python/python\\_api.html#xgboost.train](https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.train)) function in XGBoost lets us train a `Booster` with a specified set of parameters.

The code below trains a `Booster` object using a predefined labeled dataset.

```
1 # predefined data and labels
2 print('Data shape: {}'.format(data.shape))
3 print('Labels shape: {}'.format(labels.shape))
4 dtrain = xgb.DMatrix(data, label=labels)
5
6 # training parameters
7 params = {
8     'max_depth': 0,
9     'objective': 'binary:logistic'
10 }
11 print('Start training')
12 bst = xgb.train(params, dtrain) # booster
13 print('Finish training')
```



Output

3.772s

Data shape: (569, 30)

Labels shape: (569,)

Start training

[08:53:48] /workspace/src/tree/updater\_prune.cc:74: tree pruning end, 1 ro

[08:53:48] /workspace/src/tree/updater\_prune.cc:74: tree pruning end, 1 ro

[08:53:48] /workspace/src/tree/updater\_prune.cc:74: tree pruning end, 1 ro

[08:53:48] /workspace/src/tree/updater\_prune.cc:74: tree pruning end, 1 ro

[08:53:48] /workspace/src/tree/updater\_prune.cc:74: tree pruning end, 1 ro

A list of the possible parameters and their values can be found here (<https://xgboost.readthedocs.io/en/latest/parameter.html>). In the example above, we set the `'max_depth'` parameter to `0` (which means no limit on

the tree depths, equivalent to `None` in scikit-learn). We also set the `'objective'` parameter (the objective function) to binary classification via logistic regression. For the remaining available parameters, we used their default settings (so we didn't include them in `params`).

## B. Using a **Booster**

After training a **Booster**, we can evaluate it and use it to make predictions.

```
1 # predefined evaluation data and labels
2 print('Data shape: {}'.format(eval_data.shape))
3 print('Labels shape: {}'.format(eval_labels.shape))
4 deval = xgb.DMatrix(eval_data, label=eval_labels)
5
6 # Trained bst from previous code
7 print(bst.eval(deval)) # evaluation
8
9 # new_data contains 2 new data observations
10 dpred = xgb.DMatrix(new_data)
11 # predictions represents probabilities
12 predictions = bst.predict(dpred)
13 print('{}\n'.format(predictions))
```



Output

2.267s

```
Data shape: (119, 30)
Labels shape: (119,)
[0]      eval-error:0.226891
[0.6236573 0.6236573]
```

The evaluation metric used for binary classification (`eval-error`) represents the classification error, which is the default `'eval_metric'` parameter for binary classification **Booster** models.

Note that the model's predictions (from the `predict` function) are probabilities, rather than class labels. The actual label classifications are just the rounded probabilities. In the example above, the `Booster` predicts classes of 0 and 1, respectively.

## Time to Code!

The coding exercise for this chapter will be to train a `Booster` object on input `data` and `labels` (predefined in the backend).

The first thing to do is set up a `DMatrix` for training.

**Set `dtrain` equal to `xgb.DMatrix` initialized with `data` as the required argument and `labels` as the `label` keyword argument.**

1 # CODE HERE

Solution

```
1 dtrain = xgb.DMatrix(data, label=labels)
2
3
4
```

The input dataset contains 3 classes, so we'll perform multiclass classification with the `Booster`. The dataset is also relatively small, so we limit the decision tree's maximum depth to 2.

This means that the parameters for the `Booster` object will have `'max_depth'` set to 2, `'objective'` set to `'multi:softmax'`, and `'num_class'` set to 3.

**Set `params` equal to a dictionary with the specified keys and values.**

1 # CODE HERE





Solution








```
1  params = {  
2      'max_depth': 2,  
3      'objective': 'multi:softmax',  
4      'num_class': 3  
5  }  
6  
7  
8
```

Using the data matrix and parameters, we'll train the Booster .



**Set bst equal to xgb.train applied with params and dtrain as the required arguments.**

1 # CODE HERE





Solution



```
1  bst = xgb.train(params, dtrain)  
2  
3  
4
```



 Back

Introduction

Next 

Cross-Validation

 Mark as Completed

- 
-  Report an Issue
-  Ask a Question  
([https://discuss.educative.io/tag/xgboost-basics\\_\\_gradient-boosting-with-xgboost\\_\\_machine-learning-for-software-engineers](https://discuss.educative.io/tag/xgboost-basics__gradient-boosting-with-xgboost__machine-learning-for-software-engineers))