XGBTune Documentation

Release 1.1.0

R. Picard

Feb 14, 2020

Contents:

1 Get Started **1**

2 Parameters **3**
- 2.1 Configuring cross-validation . . . 3
- 2.2 Using a validation set . . . 3
- 2.3 Providing your own hyper-parameter search space . . . 4

3 Reference **5**

4 Indices and tables **7**

Python Module Index **9**

Index **11**

CHAPTER 1

Get Started

XGBTune uses a default configuration that should work in most cases. This allows to tune a model by using cross compilation‡ very easily:

```
from xgbtune import tune_xgb_model

params, round_count = tune_xgb_model(params, x_train, y_train)
```

params contains your xgboost configuration and initial parameters. x_train and y_train are your datasets with labels. The result is a tuple with the tuned parameters and the optimal number of booster rounds.

CHAPTER 2

Parameters

2.1 Configuring cross-validation

The following parameters are directly forwarded to the xgboost.cv function call:

- nfold: number of folds
- stratified: Perform stratified sampling
- folds: Sklearn KFolds or StratifiedKFolds object
- shuffle: shuffle data

These parameters must be provided as keyword parameters. Here is an example of tuning with 5 folds and without shuffling:

```
from xgbtune import tune_xgb_model

params, round_count = tune_xgb_model(
    params,
    x_train, y_train,
    folds=5, shuffle=False)
```

2.2 Using a validation set

It is possible to use a validation set instead of cross-validation:

```
from xgbtune import tune_xgb_model

params, round_count = tune_xgb_model(
    params,
    x_train, y_train,
    x_val, y_val)
```

2.3 Providing your own hyper-parameter search space

The tune_params is a dict that can be used to overload the default search space for each parameter. A search space is provided as a list, with the dict key name corresponding to the xgboost parameter: max_depth, min_child_weight, gamma, subsample, colsample_bytree, alpha, lambda, seed.

Here is an example of tuning with custom alpha search space:

```
from xgbtune import tune_xgb_model

params, round_count = tune_xgb_model(
    params,
    x_train, y_train,
    {"alpha": [(i/10.0,) for i in range(0,11)]})
```

CHAPTER 3

Reference

`xgbtune.tune.`**`tune_xgb_model`**(*params*, *x_train*, *y_train*, *x_val=None*, *y_val=None*, *nfold=3*, *stratified=False*, *folds=None*, *shuffle=True*, *tune_params={}*, *max_round_count=5000*, *loss_compare=<built-in function lt>*, *pass_count=2*, *verbose=True*)

Tunes a XGBoost model

Examples

```
>>> params, round_count = tune_xgb_model(x, y, x_val, y_val, model_params)
```

Parameters

- **params** – A dictionary with the base xgboost parameters to use
- **x_train** – Train set
- **y_train** – Train labels
- **x_val** – Validation set
- **y_val** – Validation labels
- **nfold** – Number of folds for cv
- **stratified** – Perform stratified sampling
- **folds** – Sklearn KFolds or StratifiedKFolds object
- **shuffle** – shuffle data on cross validation
- **tune_params** – dictionary containing list of values to test to each parameter
- **max_round_count** – Maximum number of rounds during training
- **pass_count** – Number of tuning pass to do

Returns A tuple of tuned parameters and round count.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

X

xgbtune.tune, 5

Index

T

`tune_xgb_model()` (*in module xgbtune.tune*), 5

X

`xgbtune.tune` (*module*), 5