TSBenchmark

Release 0.2.3

Zetyun.com

Jul 29, 2022

CONTENTS

CHAPTER

ONE

TSBENCHMARK IS A DISTRIBUTED BENCHMARK FRAMEWORK SPECIFIED FOR TIME SERIES FORECASTING TASKS USING AUTOMATED MACHINE LEARNING (AUTOML) ALGORITHMS.

TSBenchmark supports both time series and AutoML characteristics.

As for time series forecasting, it supports univariate forecasting, multivariate forecasting, as well as covariate benchmark. During operation, it collects the information of optimal parameter combinations, performance indicators and other key parameters, supporting the analysis and evaluation of the AutoML framework.

This benchmark framework supports distributed operation mode and shows high scores in efficiency ranking. It integrates the lightweight distributed scheduling framework in hypernets and can be executed in both Python and CONDA virtual environments. For the purpose of environment isolation, it is recommended to use CONDA as the environment manager to support different algorithms.

1.1 Content:

1.1.1 Concepts

Dataset

`Dataset` includes the data and metadate used in benchmark execution process. They can be obtained by the `get_train` and `get_test` functions of `TsTask` for training and testing tasks respectively.

The benchmark framework will download the dataset from cloud for the first time and save them to a cache directory for future use. The cache directory could be configured in file `benchmark.yaml`.

Task

`Task` means the training or testing tasks in `Benchmark`. They are used in `Player`. Tasks can be obtained by the `get_task` and `get_local_task` of the `tsbenchmark.api`.

`Task` consists of the following information:

- datainclude training data and testing data
- metadatainclude task type, data structure, horizon, time series field list, covariate field list, etc.
- training parametersinclude random_statereward_metricmax_trials, etc.

Player

`Player` is to run tasksA player contains a Python script file and an operating environment description file. The Python script file could call functions from TSBenchmark api to obtain the dataset, specified task, training model, evaluation methods and so on.

Benchmark

`Benchmark` makes the `Player` performing specified `Task` and integrates the results into one `Report`. These results have differences in running time, evaluation scores, etc.

TSBenchmark currently supports two kinds of Benchmark implementation

- LocalBenchmark: running Benchmark in local mode
- RemoteSSHBenchmark: running benchmark in remote mode through SSH

Environment

The operating environment of player can be either custom Python environment or virtual Python environment which are defined by the `requirement.txt` or `.yaml` file exported by conda respectively.

Report

`Report` is the valuable output of the `Benchmark`, It collects the results from players and generates a comparison report, which contains the comparison results of both different players same benchmark and same player different benchmarks.

The results include the forecast results and the performance indicators, such as smape, mae, rmse, mape, etc.

1.1.2 Quick Start

Install `tsbenchmark` with command `pip`:

```
pip install tsbenchmark
```

The document describes how to define a player and run the benchmark. An example of training a prophet model task is shown below.

1.Firstly, create a directory named `prophet_player`. Then create the subfile `player.yaml` which describes the way to build an operating environment. The example below says conda virtual environment:

```
env:
  venv:
    kind: conda  # use conda to create a virtual environment
  requirements:
    kind: conda_yaml  # use the yaml file generated by conda to configure the virtual␣
↪environment
    config:
      file_name: env.yaml

tasks:  # state that the player only supports univariate forecast task
  - univariate-forecast
```

2.Create another subfile `env.yaml` which defines the configurations of the virtual environment, under directory `prophet_player`.

```
name: tsb_prophet_player
channels:
  - defaults
  - conda-forge
dependencies:
  - prophet
  - pip:
      - tsbenchmark
```

3.Create the third subfile `exec.py` under directory `prophet_player` to perform the training task.

```
from prophet import Prophet

import tsbenchmark as tsb
import tsbenchmark.api


def main():
    task = tsb.api.get_task()
    print(task)
    m = Prophet()
    m.fit(df)
    future = m.make_future_dataframe(periods=365)
    report_data = {'reward': 0.7}
    tsb.api.report_task(report_data=report_data)

if __name__ == "__main__":
    main()
```

NOTE: The operating environment of players are created by `conda`, please firstly install conda to `/opt/miniconda3`.

4.Create the configuration file `benchmark.yaml` parallel to the main directory:

```
name: 'benchmark_example'
desc: 'local benchmark run prophet'

kind: local

players:
  - ./prophet_player

datasets:
  filter:
    tasks:
      - univariate-forecast
    data_sizes:
      - small

random_states: [ 23163 ]

constraints:
  task:
    reward_metric: rmse

venv:
  conda:
    home: /opt/miniconda3
```

So far, the directory structure looks like below:

```
.
├── benchmark.yaml
└── prophet_player
    ├── env.yaml
```

(continues on next page)

(continued from previous page)

```
    ├── exec.yaml
    └── player.yaml
```

5.Run this benchmark by the command below:

```
$ tsb run --config ./benchmark.yaml
```

When the benchmark execution ends, an experiment report is generated under directory `./report`.

1.1.3 Release Notes

Examples:

Custom Player Examples

A player generally contains a Python script `exec.py` and a Python environment description file `player.yaml`. The directory structure of a player looks like below:

```
.
├── exec.py
└── player.yaml
```

- `exec.py` is used to read tasks, train tasks and evaluate indicators according to the API provided by TSBenchmark. For more information, please refer to the documentation tsbechmark api
- `player.yaml` describes the player's Python operating environment and relevant configuration information

A comprehensive example of how to define a player is described in *Quick start*. Besides, TSBenchmark have packaged some algorithms into players. please refer to Player list.

Custom Player Operating Environment

It's possible to run `exce.py` file in the user-defined Python environment. In this case, user needs to set the argument `env.venv.kind` as `custom_python`, and put the Python executable file path after `env.venv.config.py_executable`.

`player.yaml` configuration example

```
env:
  venv:
    kind: custom_python  # set the environment as custom Python environment
  config:
    py_executable: /usr/bin/local/python  # set the Python executive file path;␣
↪otherwise use the default path
```

Define Operating Environment with `requirement.txt`

Player could use the pip dependent file requirement.txt to define the operating environment, which states all dependent packages. In this case, set the virtual environment as `conda` and set the dependency file format as `requirements_txt`. Then, benchmark will run in the virtual environment created by conda and install the dependent packages by pip.

`player.yaml` configuration example

```yaml
env:
  venv:
    kind: conda  # use conda to manage the virtual environment
    config:
      name: plain_player_requirements_txt  # name of the virtual environment
  requirements:
    kind: requirements_txt  # define dependency file format as `requirements_txt`
    config:
      py_version: 3.8  # define python version
      file_name: requirements.txt  # file name
```

The file `requirements.txt` contains the information of dependent packages in the Python virtual environment.

```
tsbenchmark
numpy >=0.1
```

Define Operating Environment with `.yaml`

`conda` could export virtual environment to a `.yaml` file which can also be used to define the player's Python virtual environment. The export method is written in Sharing an environment,

`player.yaml` configuration example

```yaml
env:
  venv:
    kind: conda  # use conda to manage the virtual environment
  requirements:
    kind: conda_yaml  # use conda_yaml to manage the dependency packages
    config:
      file_name: env.yaml  # conda yaml file name
```

NOTE: The virtual environment name has been included in the yaml file. Therefore, there is no need to define via `env.venv.config.name`.

`env.yaml` contains the information of dependent packages in the Python virtual environment.

```yaml
name: plain_player_conda_yaml
channels:
  - defaults
dependencies:
  - pip
  - pip:
      - tsbenchmark
```

Define Task Types

tsbenchmark currently supports univariate and multivariate forecast tasks. The custom player could define the task type via the argument tasks.

```
env:
  venv:
    kind: custom_python
tasks:  # define the task type
  - univariate-forecast  # options: univariate-forecast, multivariate-forecast
```

Define Randomness

If the algorithms have randomness, the benchmark is able to assign trial times to each task in order to improve the robustness and accuracy. If the algorithms have no randomness, the benchmark will run only once. The example of setting randomness is shown below:

```
env:
  venv:
    kind: custom_python
random:  false # default is true. false means the player has no randomness
```

Benchmark Examples

TSBenchmark provides the command tsb to control every benchmark. The example below shows how to use tsb to run a benchmark configured by a .yaml file:

```
$ tsb run --config <benchmark_config_file>
```

Run Benchmark in Local Mode

Local mode means executing tasks in the local machine by setting the argument kind as local.

```
name: 'benchmark_example_local' # name of benchmark
desc: 'a local benchmark example'

kind: local  # local mode

players:  # define the directory of the player
  - players/hyperts_dl_player

random_states: [ 23163,5318,9527,33179 ] # multiple runs with different random states
```

Configure the Conda Installation Directory in Local Mode

When using the conda virtual environment, it needs to configure the conda installation directory. By setting the argument `venv.conda.home` to the directory `/opt/miniconda3`, the benchmark will locate the directory and install the virtual environment accordingly.

```
name: 'benchmark_example_local' # name of benchmark
desc: 'a local benchmark example'

kind: local  # local mode

players:
  - players/hyperts_dl_player

datasets:
  tasks_id:
    - 512754

random_states: [ 23163,5318,9527,33179 ]

venv:
  conda:
    home: /opt/miniconda3  # define the conda installation directory
```

Select tasks

User could set different filtering conditions to select desired tasks. Benchmark provides the three types of conditions and each condition has multiple options.

- `datasets.filter.tasks`: select one or more task types, default (empty) means all types
- `datasets.filter.data_sizes`: select one or more data size types (small/large), default (empty) means all types
- `datasets.filter.tasks_id`: define the task identifications

An example of tasks selection

```
name: 'benchmark_example_local'
desc: 'a local benchmark example'

kind: local

players:
  - players/hyperts_dl_player

datasets:
  filter:
    tasks:  # select two task types: univariate-forecastmultivariate-forecast
      - univariate-forecast
      - multivariate-forecast
    data_sizes:  # select small-size dataset
      - small
    tasks_id:  # define task id
```

(continues on next page)

(continued from previous page)

```
      - 512754

random_states: [ 23163, 5318,9527,33179 ]
```

Configure Constraints

Benchmark could set some constraints like the trial times of algorithm searching (`max_trials`), evaluation indicator definition (`reward_metric`) and so on.

```
name: 'benchmark_example_local'
desc: 'a local benchmark example'

kind: local

players:
  - players/hyperts_dl_player

random_states: [ 23163,5318,9527,33179 ]

constraints:  # configure the constraints
  task:
    max_trials: 10
    reward_metric: rmse
```

Multiple Runs with Defined Random States

TSBenchmark could set player to run the same task with multiple defined random states, which could help to reduce the randomness and evaluate the stability of algorithms. The random states and its number could be set by the argument `random_states` and `n_random_states`. See example below

```
name: 'benchmark_example_local'
desc: 'a local benchmark example'

kind: local

players:
  - players/hyperts_dl_player

random_states: [ 23163,5318,9527,33179 ]  # the task runs 4 times with these 4 random␣
↪states.
```

Run Benchmark in Remote (Multi-machine) Mode

Running benchmark in remote/multi-machine mode could speed up the execution process.It requires TSBenchmark assign tasks to multiple notes by SSH protocol. First, set the argument `kind` as `remote` and then configure the argument `machines` with remote connection information. If the player requires virtual operating environment, the remote machine needs to install conda and specify the conda installation directory.

```
name: 'benchmark_example_remote'
desc: 'a remote benchmark example'

kind: remote

players:
 - players/hyperts_dl_player

random_states: [ 23163,5318,9527,33179 ]

machines:  # remote machine, SSH
 - connection:  # remote machine information
       hostname: host1
       username: hyperctl
       password: hyperctl
   environments:
     TSB_CONDA_HOME: /opt/miniconda3  # specify the conda installation directory
```

Rerun Benchmark

When rerunning a benchmark, the previous executed tasks (either failed or successful) will be skipped. The state files of both failed and successful tasks are shown below. To rerun executed tasks, user could delete the corresponding state files.

- State file of successful task`{working_dir}/batches/{benchmark_name}/{job_name}.succeed`
- State file of failed state`{working_dir}/batches/{benchmark_name}/{job_name}.failed`

The output data and state information of an executed benchmark will be written under the directory `working_dir`. If executing a benchmark that is continuous of another, make sure the configurations of `working_dir` and `name` of the two benchmarks are consistent

```
name: 'benchmark_example_local' # name of benchmark
desc: 'a local benchmark example'

kind: local

working_dir: /tsbenchmark_working_dir/benchmark_example_local  # the directory of␣
↪Benchmark which store the output files; default(empty) dir `/tsbenchmark_working_dir`

players:
  - players/hyperts_dl_player
```

1.1.4 tsbenchmark

tsbenchmark package

tsbenchmark.api module

tsbenchmark.api.**get_local_task**(*data_path*, *dataset_id='512754'*, *random_state=2022*, *max_trials=3*, *reward_metric='smape'*) → *TSTask*

Get a TsTask from local for develop a new player and test.

TsTask is a unit task, which help Player get the data and metadata. It will get a TsTaskConfig locally and construct it to TSTask. Call TSTask.ready() method init start time and load data.

Parameters

- **data_path** – str, default='~/tmp/data_cache'. The path locally to cache data. TSLoader will download data and cache it in data_path.
- **dataset_id** – str, default='512754'. The unique id for a dataset task. You can get it from tests/dataset_desc.csv.
- **random_state** – int, consts.GLOBAL_RANDOM_STATE. Determines random number for automl framework.
- **max_trials** – int, default=3. Maximum number of tests for automl framework, optional.
- **reward_metric** – str, default='smape'. The optimize direction for model selection. Hypernets search reward metric name or callable. Possible values: 'accuracy', 'auc', 'mse', 'mae','rmse', 'mape', 'smape', and 'msle'.

Notes

1. You can get attributes description from TSTask.
2. In the report it support 'smape', 'mape', 'mae' and 'rmse'.

See also:

TSTask: Player will get the data and metadata from the TSTask then run algorithm for compete.

Returns: TSTask, The TsTask for player get the data and metadata.

tsbenchmark.api.**get_task**() → *TSTask*

Get a TsTask from benchmark server.

TsTask is a unit task, which help Player get the data and metadata. It will get TsTaskConfig from benchmark server and construct it to TSTask. Call TSTask.ready() method init start time and load data.

See also:

TSTask : Player will get the data and metadata from the TSTask then run algorithm for compete.

Notes

1. You can get attributes description from TSTask.
2. In the report it support 'smape', 'mape', 'mae' and 'rmse'.

Returns: TSTask, The TsTask for player get the data and metadata.

tsbenchmark.api.**report_task**(*report_data: Dict*, *bm_task_id=None*, *api_server_uri=None*)

Report metrics or running information to api server.

Parameters

- **report_data** – Dict. The report data generate by send_report_data.
- **bm_task_id** – str, optional, BenchmarkTask id, if is None will get from current job
- **api_server_uri** – str, optional, tsbenchmark api server uri, if is None will get from environment or use default value

tsbenchmark.api.**send_report_data**(*task: TSTask*, *y_pred: DataFrame*, *key_params=''*, *best_params=''*, *sub_result=False*)

Send report data.

This interface used for send report data to benchmark server. 1. Prepare the data which can be call be tsb.api.report_task. 2. Call method report_task, send the report data to the Benchmark Server.

Parameters

- **y_pred** – pandas.DataFrame, The predicted values by the players. It should be a pandas.DataFrame, and it must have the headers name, which you can get from task.series_name.
- **key_params** – str, default=" The params which user want to save to the report datas.
- **best_params** – str, default=" The best model's params, for automl, there are many models will be trained. If user want to save the best params, user may assign the best_params.

Notes

When develop a new play locally, this method will help user validate the predicted and params.

tsbenchmark.tasks module

class tsbenchmark.tasks.**TSTask**(*task_config*, ***kwargs*)

Bases: `object`

Player will get the data and metadata from the TSTask then run algorithm for compete.

Parameters

- **dataset_id** – str, not None. The unique identification id.
- **date_name** – str, not None. The name of the date column.
- **task** – str, not None. The type of forecast. In time series task, it could be 'univariate-forecast' or 'multivariate-forecast'.
- **horizon** – int, not None. Number of periods of data to forecast ahead.
- **shape** – str, not None. The dataset shape from the train dataframe. The result from pandas.DataFrame.shape().

- **series_name** – str or arr. The names of the series columns. For 'univariate-forecast' task, it should not be None.For 'multivariate-forecast' task, it should be None. In the task from tsbenchmark.api.get_task() or tsbenchmark.api.get_local_task or called function TSTask.ready, series_name should not be None.
- **covariables_name** – str or arr, may be None. The names of the covariables columns. It should be get after called function TSTask.ready(), or from task from tsbenchmark.api.get_task() or tsbenchmark.api.get_local_task.
- **dtformat** – str, not None. The format of the date column.
- **random_state** – int, consts.GLOBAL_RANDOM_STATE Determines random number for automl framework.
- **max_trials** – int, default=3. Maximum number of tests for automl framework, optional.
- **reward_metric** – str, default='smape'. The optimize direction for model selection. Hypernets search reward metric name or callable. Possible values: 'accuracy', 'auc', 'mse', 'mae','rmse', 'mape', 'smape', and 'msle'.

Notes

In the report it support 'smape', 'mape', 'mae' and 'rmse'.

`get_data()`

Get data contain train_data and test_data which will be used in the Player.

`get_test()`

Get a pandas.DadaFrame test data which will be used in the Player.

Returns
The data for test.

Return type
pandas.DataFrame

`get_train()`

Get a pandas.DadaFrame train data which will be used in the Player.

Returns
The data for train.

Return type
pandas.DataFrame

`ready()`

Init data download if the data have not been download yet.

`to_dict()`

1.1.5 Release Notes

History:

Version 0.1.0

This version has the following features

Dataset

- Univariate dataset
- Multivariate dataset
- Support corvariate
- Data download

Task

- Univariate forecast
- Multivariate forecast
- Task selection

Operation mode

- Distributed operation
- Pseudo-distributed operation
- Breakpoint continuation
- Command tool

Environment management

- Environment isolation
- Default Python environment
- Environment setup

Information acquisition

- Performance indicators
- Time consuming
- Optimized parameters
- Key parameters

Report

- Performance comparison
- Time consuming comparison
- Random states comparison
- Versions comparison

Packaged Players

- HyperTS(STAT & DL)
- Pyaf

- Autots
- Fedot
- Navie & SNavie

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

TSBenchmark is an open source project created by DataCanvas .

PYTHON MODULE INDEX

t

INDEX