TSBenchmark

zetyun.com

2022 年 07 月 29 日

Contents

CHAPTER 1

TSBenchmark: 一个面向时间序列预测自动机器学习算法的分布式 Benchmark 框架

TSBenchmark 同时支持 Time Series 特性与 AutoML 特性。时间序列预测算法，支持单变量预测与多变量预测，同时支持协变量 benchmark。运行过程中，支持最优参数组合采集，为 AutoML 框架的分析提供支撑。

框架支持分布式运行模式，具备高效的跑评分效率。框架集成了 Hypernets 的内的轻量级分布式调度框架,python 或者 conda 环境下均可运行。推荐使用 conda 作为环境管理，以支持不同时序算法的环境隔离。

1.1 内容:

1.1.1 概念

Dataset

`Dataset` 是供 `Player` 跑 benchmark 时使用的数据和元数据, 通过 `TsTask` 对象的 get_train 和 get_test 方法可以获取。

初次运行时，框架自动从云端下载数据集。已下载成功的数据集会统一保存到缓存目录，后续运行时会使用缓存数据。数据缓存目录地址可以在 `benchmark.yaml` 中指定。

Task

`Task` 是 `Benchmark` 的评测的一个原子任务。其主要供 `Player` 中使用，用户可以通过 tsbenchmark.api 的 get_task 和 get_local_task 获取。

`Task` 包含几部分信息

- 数据，包括训练数据和测试数据
- 元数据，包括任务类型、数据形状、horizon、时间序列字段列表、协变量字段列表等
- 训练参数，包括 random_state、reward_metric、max_trials 等 `Benchmark` 参数

Benchmark

`Benchmark` 通过使一组 `Player` 分别运行一组相同的 `Task`，并将运行的结果汇总成一个 `Report`。在 Report 中包含这些 `Player` 运行任务消耗的时间、评估指标得分等信息。

目前有两个 Benchmark 的实现，分别是：

- LocalBenchmark: 在本地运行的 Benchmark
- RemoteSSHBenchmark: 通过 SSH 协议将训练任务分发到远程机器运行的 Benchmark，

Player

Player 用来运行 Task。它会包含一个 python 运行环境和一个 python 脚本文件。在 python 脚本中，可以调用 tsbenchmark 提供的 api 获取 Task，训练模型，评估模型和上传运行数据。

Environment

Player 的 python 运行环境。可以是自定义 python 环境，也可以是由 conda 管理的虚拟 python 环境。可以使用 `requirement.txt` 或者 conda 导出的 yaml 文件定义虚拟环境。如果是使用 conda 管理的虚拟环境，需要安装好conda，并在 Benchmark 的配置文件中设置 conda 的安装目录。Benchmark 运行时会使用 conda 创建虚拟 python 环境并用此环境运行 player 的 `exec.py`。

Report

`Report` 是供 `Benchmark` 的最终成果，收集 `Player` 的反馈结果信息，并生成对比分析报告。报告支持不同 `Player` 间的对比报告横向对比，也支持同 `Player` 跑不同的 Benchmark 之间的对比纵向对比。

报告包含预测结果归档和 Performance 对比，其中的 Performance 包含 smape, mae, rmse 和 mape 的常见评价指标.

1.1.2 快速开始

使用 pip 安装 tsbenchmark:

```
$ pip install tsbenchmark
```

以使用prophet训练为例定一个 player，创建目录 `prophet_player`，并在该目录中创建 `player.yaml` 文件，内容为:

```
env:
  kind: conda
  requirements:
    kind: conda_yaml
```

(续下页)

(接上页)

```
    config:
      file_name: env.yaml

tasks:
  - univariate-forecast
```

接着在目录 prophet_player 中创建 env.yaml 文件，这个文件用来使用conda创建虚拟环境，在 player 运行时使用，文件内容为：

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

最后，在目录 prophet_player 中创建 exec.py 文件用来使用 prophet 训练任务，文件内容为：

```
from prophet import Prophet

import tsbenchmark as tsb
import tsbenchmark.api


def main():
    # task = tsb.api.get_local_task(data_path="/tmp/hdatasets", dataset_id=512754,␣
↪random_state=9527, max_trials=1, reward_metric='rmse')
    task = tsb.api.get_task()
    df_train = task.get_train().copy(deep=True)
    df_train.rename(columns={task.date_name: 'ds', task.series_name[0]: 'y'},␣
↪inplace=True)

    df_test = task.get_test().copy(deep=True)
    df_test.rename(columns={task.date_name: 'ds', task.series_name[0]: 'y'},␣
↪inplace=True)

    model = Prophet()
    model.fit(df_train)

    df_prediction = model.predict(df_test)  # 评估模型
```

(续下页)

(接上页)

```
    df_result = df_prediction[['yhat']].copy(deep=True)
    df_result.rename(columns={'yhat': task.series_name[0]}, inplace=True)

    tsb.api.send_report_data(task=task, y_pred=df_result)  # 上报评估数据


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

由于 player 的运行环境需要使用使用conda创建，请先安装 conda 到/opt/miniconda3，如安装到其他目录请配置 venv.conda.home 为您的 conda 安装目录。

然后在当前目录创建 Benchmark 配置文件 benchmark.yaml:

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

kind: local

players:
  - ./prophet_player

tasks:
  filter:
    task_ids:
      - '512754'

random_states: [ 23161, 23162, 23163 ]

constraints:
  task:
    reward_metric: rmse

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

当前目录的结构应该为:

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

(续下页)

(接上页)

```
    └── player.yaml
```

运行该 benchmark:

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

运行结束后可以去 `~/tsbenchmark-data-dir/benchmark_example/report` 目录查看生成的测试报告。

1.1.3 配置文件参考

Player 配置文件参考

Player 通常会包含一个 yaml 格式的描述文件 `player.yaml` 和一个 python 脚本 `exec.py`，一个 player 的目录结构看起来像是：

```
.
├── exec.py
└── player.yaml
```

- `exec.py` 脚本来借助 tsbenchmark 提供的 api 完成读取任务、训练任务、和评估指标，api 用法参考*TSBenchmark API References* 。
- `player.yaml` 用来描述 player 的配置信息。

定义 player 例子请参考快速开始，在 TSBenchmark 中也已经将一些算法封装成 Player，参考 Player 列表 。

配置样例

自定义 python 环境

```
name: hyperts_player
env:
  kind: custom_python  # 使用自定义python环境
  py_executable: /usr/bin/python
```

conda 管理 conda 格式依赖文件

```
name: hyperts_player
env:
  kind: conda  # 使用conda创建虚拟环境
  requirements:
    kind: conda_yaml # 使用conda 格式的依赖
    file_name: env.yaml
```

conda 管理 pip 格式依赖文件

```
name: hyperts_player
env:
  kind: conda  # 使用conda创建虚拟环境
  requirements:
    kind: requirements_txt # 使用pip格式的依赖定义文件
    file_name: requirements.txt
    py_version: 3.8

tasks:  # 仅支持单变量预测的任务
  - univariate-forecast

random: true  #  使用随机数
```

配置项参考

PlayerConfig

Field Name	Type	Description
name	`str`, optional	player 的名称，如果为空将使用 player 所在的文件夹名。
env	*EnvConfig*, required	运行环境配置。
tasks	`list[str]`, optional	player 支持的任务类型，默认为空，如果为空表示支持所有任务类型。 Benchmark 运行时只会给 Player 分配它能支持的类型的任务; 可选的值有 `univariate-forecast`, `multivariate-forecast`。
random	`boolean`, optional	player 是否接受随机数，默认为 `true`。 如果接受随机数，Benchmark 运行时会对每个任务使用不同的随机数跑多次减少随机因素带来的影响。 如果不接受则仅运行任务一次。

EnvConfig

是下列对象的一种：

- *CustomPythonEnvConfig*
- *CondaEnvConfig*

CustomPythonEnvConfig

使用已经创建好的 python 环境运行 player。这种情况下 benchmark 运行时候不会再为 player 创建虚拟环境，而是使用指定的 python 环境运行。

Field Name	Type	Description
kind	`"custom_python"`	
py_executable	`str`, optional	python 的可执行文件路径，默认为当前进程使用的 python。

CondaEnvConfig

定义使用 conda 管理的虚拟环境。Benchmark 运行时候会使用已经配置好的 conda 创建虚拟环境并运行 player。

Field Name	Type	Description
kind	`"conda"`	
name	`str`, optional	conda 虚拟环境的名称。如果为空，当 `env.requirements.kind=requirements_txt` 时使用 player 的 name; 当 `env.requirements.kind=conda_yaml` 时使用 conda 环境的 yaml 文件中的名称。
requirements	*RequirementsConfig*	定义虚拟环境的依赖包。

备注： 如果运行时根据虚拟环境的名称检查到虚拟环境已经存在则会跳过环境创建并使用当前存在的环境运行 player。

RequirementsConfig

是下列对象的一种：

- *RequirementsTxtConfig*
- *CondaYamlConfig*

RequirementsTxtConfig

player 可以使用 pip 的依赖文件格式 (requirement.txt) 声明所需要的依赖库，一个 `requirement.txt` 文件看起来像：

```
tsbenchmark
numpy >=0.1
```

benchmark 运行时候会使用 conda 创建虚拟环境并使用 pip 安装依赖。

Field Name	Type	Description
kind	`"requirements_txt"`	
py_version	`str`, optional	虚拟环境的 python 版本，如果为空将使用当前进程使用的 python 版本。
file_name	`str`, optional	pip 依赖文件的名称，默认为 `requirements.txt`, 此文件存放在 player 目录中。由于 player 运行时候需要使用 tsbenchmark，请在该文件中添加 tsbenchmark。

CondaYamlConfig

conda 可以将虚拟环境导出成 yaml 文件，参考 Sharing an environment 。导出的文件看起来像：

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

导出的 yaml 文件可以用来定义 player 的虚拟 python 环境，Benchmark 运行时候会使用此文件创建虚拟环境并用来运行 Player。

Field Name	Type	Description
kind	`"conda_yaml"`	
file_name	`str`, optional	conda 虚拟环境导出的 yaml 文件，默认为 `env.yaml`, 此文件存放在 player 目录中。此文件中通常已经包含虚拟环境的名称，不必再通过 `env.name` 配置虚拟环境的名称；

Benchmark 配置文件参考

tsbenchmark 提供了命令行工具 `tsb` 命令管理 Benchmark。可以使用 yaml 格式的配置文件定义 benchmark，并使用 tsb 命令运行：

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

配置样例

LocalBenchmark

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

kind: local  # 单机模式

players:
  - players/hyperts_dl_player

tasks:
  filter:
    task_ids:
      - '512754'

random_states: [ 23163, 5318, 9527 ]

venv:
  conda:
    home: /opt/miniconda3  # 配置本机conda安装位置
```

RemoteSSHBenchmark

```yaml
name: 'benchmark_example_local'
desc: 'a remote benchmark example'

kind: remote  # 远程并行模式

players:
  - players/hyperts_dl_player

tasks:
  filter:
    task_ids:
      - '512754'

random_states: [ 23163, 5318, 9527 ]

machines:
  - connection: # 配置远程SSH机器连接方式
```

(续下页)

(接上页)

```
        hostname: host1
        username: hyperctl
        password: hyperctl
    environments: # 配置远程SSH机器conda安装位置
      TSB_CONDA_HOME: /opt/miniconda3

batch_application_config:
  server_port: 8060
  server_host: 192.168.300.300  # 可以被远程机器访问的ip地址
```

配置项参考

BaseBenchmarkConfig

Benchmark 有两个实现：

- LocalBenchmark: 单机模式运行的 Benchmark，专用配置见*LocalBenchmarkConfig*
- RemoteSSHBenchmark: 基于 SSH 协议并行运行的 Benchmark, 专用配置见*RemoteSSHBenchmarkConfig*

这部分配置是这两个实现的通用的。

Field Name	Type	Description
name	`str`, required	benchmark 的名称，可以使用数字、大小写字母、下划线、中划线组合。
desc	`str`, optional	Benchmark 描述。
kind	`str`, required	Benchmark 的类型，可选 `local` 和 `remote` ，分别对应 `LocalBenchmark` 和 `LocalBenchmark` 的 Benchmar 实现。
benchmarks_data_dir	`str`, optional	用于存放 Benchmark 运行产生的文件; 默认为 `~/tsbenchmark-data-dir`，将会以 Benchmark 的 name 为名称为每个 Benchmark 在此目录下创建子目录。
players	`list[str]`, required	Benchmark 使用到的 Player 的本地目录地址。如果是 `RemoteSSHBenchmark` 这些目录将会被上传到远程机器使用。
constraints	*ConstraintsConfig*, required	运行 Benchmark 的约束条件。
batch_application_config	*BatchApplicationConfig*, optional	配置 Hyperctl。
tasks	*TaskConfig*, optional	设置参与 Benchmark 的任务。
random_states	`list[int]`, optional	Benchmark 运行时会让 Player 使用不同的随机数运行同一个任务，这样可以降低实验的随机性，默认为 `[9527]` 。

备注： 当一个 Benchmark 重复运行时，之前运行结束（失败或者成功状态）的任务会被跳过不再运行。如需重新运行 Benchmark 中已经结束的任务，可以删除该任务的状态文件, 任务的状态文件在：

- 任务成功的状态文件：`{benchmarks_data_dir}/{benchmark_name}/batch/{job_name}.succeed`
- 任务失败的状态文件：`{benchmarks_data_dir}/{benchmark_name}/batch/{job_name}.failed`

若要实现一次 Benchmark 基于上一次 Benchmark 运行时跳过已经结束的任务，需要确保这两次运行的 Benchmark 的 `benchmarks_data_dir` 和 `name` 属性一致。

TaskConfig

Field Name	Type	Description
cache_path	`str`, optional	下载 Dataset, Task 的缓存目录，加载数据集或任务时会优先读取缓存，如果缓存不存在再从 source 中加载。默认读取环境变量 `TSB_DATASETS_CACHE_PATH`，如果不存在使用 `~/.cache/tsbenchmark/datasets` 。
filter	*Task-Filter-Config*, optional	配置筛选任务的条件。
source	`str`, optional	数据集和任务的下载源。默认为 `AWS` 。

TaskFilterConfig

使用所有的任务运行 Benchmark 将消耗很多资源和时间，因此可以使用过滤条件指定哪些任务用来运行 Benchmark。

Field Name	Type	Description
task_types	`list[str]` optional	按任务类型筛选，默认为使用所有类型的任务。可选的值有 `univariate-forecast`, `multivariate-forecast`。
datasets_sizes	`list[str]` optional	按数据集的大小筛选, 默认选择所有大小类型的数据集文件; 可选 `small`, `large`。
task_ids	`list[str]` optional	指定任务的 id。
dataset_ids	`list[str]` optional	指定数据集的 id。

备注： 过滤条件可以指定一个或者多个, 多个筛选条件之间的是”与”的关系，如果没有设置筛选条件将使用所有任务。

ConstraintsConfig

运行 Benchmark 可以设定一些约束条件。比如设置 Player 中的算法搜索的次数、评价指标等。

Field Name	Type	Description
task	*TaskConstraintsConfig*	对任务的约束条件。

TaskConstraintsConfig

任务的约束参数在 Player 中可以接受到，player 中的算法需要使用这些参数运行任务。

Field Name	Type	Description
max_trials	`int`, optional	最大搜索次数，默认是 10。
reward_metric	`str`, optional	设置调参的评价指标，默认是 `rmse`。

BatchApplicationConfig

TSBenchmark 使用 Hyperctl 管理任务。

Field Name	Type	Description
server_port	`int`, optional	服务端口，默认为 `8086` 。
server_host	`str`, optional	Hyperctl 服务地址，默认为 `localhost` , 如果是并行运行模式请将该地址配置为远程节点可以访问的 ip。
scheduler_interval	`int`, optional	调度周期，默认为 `5000`, 单位毫秒。
scheduler_exit_on_finish	`boolean`, optional	所有任务结束后是否退出进程，默认为 `true` 。

LocalBenchmarkConfig

单机模式运行的 Benchmark 特有的配置，这种模式下训练任务都将在当前机器上进行，配置样例见*LocalBenchmark* 。

Field Name	Type	Description
venv	*LocalVenvConfig*	配置当前机器上的虚拟环境管理器信息。

LocalVenvConfig

Field Name	Type	Description
conda	*LocalCondaConfig*	配置 Conda 虚拟环境管理器的信息。

LocalCondaConfig

Field Name	Type	Description
home	`str`, optional	conda 的安装目录, 如果在 Benchmark 中用到的 player 有使用 conda 虚拟环境的，需要配置 conda 的安装目录。 Benchmark 在运行的时候可以使用这个 conda 创建虚拟环境。

RemoteSSHBenchmarkConfig

基于 SSH 协议并行运行的 Benchmark 特有的配置，这种模式以利用多台机器加快 Benchmark 的运行进度。它将任务通过 SSH 协议分发的远程节点，这要求远程运行任务的节点需要运行 SSH 服务，并且提供连接帐号。如果运行的 player 中有使用到 conda 创建虚拟环境的，还需要在远程机器中安装好 conda。配置样例见*RemoteSSHBenchmark* 。

Field Name	Type	Description
machines	list[*RemoteMachineConfig*], required	远程机器的的链接信息和配置信息, Benchmark 会将训练任务分发到这些节点上。

RemoteMachineConfig

Field Name	Type	Description
connection	*SHHConnectionConfig*, required	远程机器的的链接信息。

environments	`dict`, optional	远程机器的环境信息。如果运行的 Player 有使用 conda 虚拟环境的，需要通过键 `TSB_CONDA_HOME` 配置 conda 的安装目录，例如：
		`machines:`
		`  - connection:`
		`  hostname: host1`
		`  username: hyperctl`
		`  password: hyperctl`
		`  environments:`
		`  TSB_CONDA_HOME: /opt/miniconda3  # 配置conda的安装目录`

SHHConnectionConfig

Field Name	Type	Description
hostname	`hostname` required	远程机器的 ip 地址或者主机名。
username	`username` required	远程机器的用户名。
password	`password` required	远程机器的连接密码。

1.1.4 TSBenchmark API References

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.

参数

- **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'.

备注

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

参见:

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.

参见:

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

备注

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.

参数

- **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.

参数

- **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.

备注

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*)

基类：object

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

参数

- **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’.

备注

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.

返回

The data for test.

返回类型

pandas.DataFrame

get_train()

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

返回

The data for train.

返回类型

pandas.DataFrame

ready()

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

to_dict()

1.1.5 Release Notes

历史:

Version 0.1.0

本次发布新增以下特性:

数据集

- 单变量数据集
- 多变量数据集
- 数据下载
- 协变量支持

任务

- 预测（单变量 & 多变量）
- 任务过滤

运行引擎

- 分布式运行
- 伪分布式运行
- 断点续跑

- 命令行工具

环境管理

- 环境隔离
- 默认 python 环境
- 环境 setup

信息采集

- performance 指标
- 耗时
- 最优参数组合
- 核心参数

报告

- peformance 指标对比
- 耗时对比
- 多随机因子统计
- 版本对比

预置 Players

- HyperTS(STAT & DL)
- Pyaf
- Autots
- Fedot
- Navie & SNavie

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

TSBenchmark is an open source project created by DataCanvas .

Python 模块索引

t

索引