# PyHealth Documentation

*Release 0.0.6*

**Yue Zhao**

**Oct 23, 2022**

# GETTING STARTED

**[Oct 2022] We will release a brand-new version of PyHealth in the next few weeks. It will include more EHR datasets, health-related tasks, and state-of-the-art models. Please stay tuned!**

---

**Deployment & Documentation & Stats**

---

**Build Status & Coverage & Maintainability & License**

---

**PyHealth** is a comprehensive **Python package** for **healthcare AI**, designed for both **ML researchers** and **healthcare and medical practitioners**. **PyHealth** accepts diverse healthcare data such as longitudinal electronic health records (EHRs), continuous signials (ECG, EEG), and clinical notes (to be added), and supports various predictive modeling methods using deep learning and other advanced machine learning algorithms published in the literature.

The library is proudly developed and maintained by researchers from Carnegie Mellon University, IQVIA, and University of Illinois at Urbana-Champaign. PyHealth makes many important healthcare tasks become accessible, such as **phenotyping prediction**, **mortality prediction**, and **ICU length stay forecasting**, etc. Running these prediction tasks with deep learning models can be as short as 10 lines of code in PyHealth.

**PyHealth comes with three major modules**: (i) *data preprocessing module*; (ii) *learning module* and (iii) *evaluation module*. Typically, one can run the data prep module to prepare the data, then feed to the learning module for prediction, and finally assess the result with the evaluation module. Users can use the full system as mentioned or just selected modules based on the own need:

- **Deep learning researchers** may directly use the processed data along with the proposed new models.
- **Medical personnel**, may leverage our data preprocessing module to convert the medical data to the format that learning models could digest, and then perform the inference tasks to get insights from the data.

PyHealth is featured for:

- **Unified APIs, detailed documentation, and interactive examples** across various types of datasets and algorithms.
- **Advanced models**, including **latest deep learning models** and **classical machine learning models**.
- **Wide coverage**, supporting **sequence data**, **image data**, **series data** and **text data** like clinical notes.

- **Optimized performance with JIT and parallelization** when possible, using numba and joblib.
- **Customizable modules and flexible design**: each module may be turned on/off or totally replaced by custom functions. The trained models can be easily exported and reloaded for fast execution and deployment.

**API Demo for LSTM on Phenotyping Prediction with GPU**:

```
# load pre-processed CMS dataset
from pyhealth.data.expdata_generator import sequencedata as expdata_generator

expdata_id = '2020.0810.data.mortality.mimic'
cur_dataset = expdata_generator(exp_id=exp_id)
cur_dataset.get_exp_data(sel_task='mortality', )
cur_dataset.load_exp_data()

# initialize the model for training
from pyhealth.models.sequence.lstm import LSTM
# enable GPU
expmodel_id = 'test.model.lstm.0001'
clf = LSTM(expmodel_id=expmodel_id, n_batchsize=20, use_gpu=True, n_epoch=100)
clf.fit(cur_dataset.train, cur_dataset.valid)

# load the best model for inference
clf.load_model()
clf.inference(cur_dataset.test)
pred_results = clf.get_results()

# evaluate the model
from pyhealth.evaluation.evaluator import func
r = func(pred_results['hat_y'], pred_results['y'])
print(r)
```

**Citing PyHealth**:

PyHealth paper is under review at JMLR (machine learning open-source software track). If you use PyHealth in a scientific publication, we would appreciate citations to the following paper:

```
@article{zhao2021pyhealth,
  title={PyHealth: A Python Library for Health Predictive Models},
  author={Zhao, Yue and Qiao, Zhi and Xiao, Cao and Glass, Lucas and Sun, Jimeng},
  journal={arXiv preprint arXiv:2101.04209},
  year={2021}
}
```

or:

```
Zhao, Y., Qiao, Z., Xiao, C., Glass, L. and Sun, J., 2021. PyHealth: A Python Library␣
↪for Health Predictive Models. arXiv preprint arXiv:2101.04209.
```

**Key Links and Resources**:

- View the latest codes on Github
- Execute Interactive Jupyter Notebooks
- Check out the PyHealth paper

CHAPTER

ONE

# PREPROCESSED DATASETS & IMPLEMENTED ALGORITHMS

**(i) Preprocessed Datasets** (customized data preprocessing function is provided in the example folders):

| Type | Abbr | Description | Processed Function | Link |
|---|---|---|---|---|
| Sequence: EHR-ICU | MIMIC III | A relational database containing tables of data relating to patients who stayed within ICU. | \examples\data_generation\dataloader_mimic | https://mimic.physionet.org/gettingstarted/overview/ |
| Sequence: EHR-ICU | MIMIC_Demo | The MIMIC-III demo database is limited to 100 patients and excludes the noteevents table. | \examples\data_generation\dataloader_mimic_demo | https://mimic.physionet.org/gettingstarted/demo/ |
| Sequence: EHU-Claim | CMS | DE-SynPUF: CMS 2008-2010 Data Entrepreneurs Synthetic Public Use File | \examples\data_generation\dataloader_cms | https://www.cms.gov/Research-Statistics-Data-and-Systems/Downloadable-Public-Use-Files/SynPUFs |
| Image: Chest X-ray | Pediatric | Pediatric Chest X-ray Pneumonia (Bacterial vs Viral vs Normal) Dataset | N/A | https://academictorrents.com/details/951f829a8eeb4d2839c4a535db95078a9175010b |
| Series: ECG | PhysioNet | AF Classification from a short single lead ECG recording Dataset. | N/A | https://archive.physionet.org/challenge/2017/#challenge-data |

You may download the above datasets at the links. The structure of the generated datasets can be found in datasets folder:

- \datasets\cms\x_data\...csv
- \datasets\cms\y_data\phenotyping.csv
- \datasets\cms\y_data\mortality.csv

The processed datasets (X,y) should be put in x_data, y_data correspondingly, to be appropriately digested by deep learning models. We include some sample datasets under \datasets folder.

**(ii) Machine Learning and Deep Learning Models** :

**For sequence data**:

| Type | Abbr | Class | Algorithm | Year | Ref |
|---|---|---|---|---|---|
| Classical Models | RandomForest | *pyhealth.models.sequence.rf.RandomForest* | Random forests | 2000 | [ABre01] |
| Classical Models | XGBoost | `pyhealth.models.sequence.xgboost.XGBoost` | XGBoost: A scalable tree boosting system | 2016 | **[#Chen2016Xgboost]_** |
| Neural Networks | LSTM | *pyhealth.models.sequence.lstm.LSTM* | Long short-term memory | 1997 | **[#Hochreiter1997Long]_** |
| Neural Networks | GRU | *pyhealth.models.sequence.gru.GRU* | Gated recurrent unit | 2014 | **[#Cho2014Learning]_** |
| Neural Networks | RETAIN | *pyhealth.models.sequence.retain.RetainAttention* | RETAIN: An Interpretable Predictive Model for Healthcare using Reverse Time Attention Mechanism | 2016 | **[#Choi2016RETAIN]_** |
| Neural Networks | Dipole | *pyhealth.models.sequence.dipole.Dipole* | Dipole: Diagnosis Prediction in Healthcare via Attention-based Bidirectional Recurrent Neural Networks | 2017 | **[#Ma2017Dipole]_** |
| Neural Networks | tLSTM | *pyhealth.models.sequence.tlstm.tLSTM* | Patient Subtyping via Time-Aware LSTM Networks | 2017 | **[#Baytas2017tLSTM]_** |
| Neural Networks | RAIM | *pyhealth.models.sequence.raim.RAIM* | RAIM: Recurrent Attentive and Intensive Model of Multimodal Patient Monitoring Data | 2018 | **[#Xu2018RAIM]_** |
| Neural Networks | StageNet | *pyhealth.models.sequence.stagenet.StageNet* | StageNet: Stage-Aware Neural Networks for Health Risk Prediction | 2020 | **[#Gao2020StageNet]_** |

**For image data**:

**For ecg/egg data**:

| Type | Abbr | Class | Algorithm | Year | Ref |
|---|---|---|---|---|---|
| Classical Models | RandomForest | pyhealth.models.ecg.rf | Random Forests | 2000 | [#Breiman2001Random]_ |
| Classical Models | XGBoost | pyhealth.models.ecg.xgboost | XGBoost: A scalable tree boosting system | 2016 | [#Chen2016Xgboost]_ |
| Neural Networks | BasicCNN1D | pyhealth.models.ecg.approach | Face recognition: A convolutional neural-network approach | 1997 | [#Lawrence1997Face]_ |
| Neural Networks | DBLSTM-WS | pyhealth.models.ecg.dblstm_ws | A novel wavelet sequence based on deep bidirectional LSTM network model for ECG signal classification | 2018 | |
| Neural Networks | DeepRes1D | pyhealth.models.ecg.deepres1d | Heartbeat classification using deep residual convolutional neural network from 2-lead electrocardiogram | 2019 | |
| Neural Networks | AE+BiLSTM | pyhealth.models.ecg.aebilstm | Automatic Classification of CAD ECG Signals With SDAE and Bidirectional Long Short-Term Network | 2019 | |
| Neural Networks | KRCRnet | pyhealth.models.ecg.krcrnet | K-margin-based Residual-Convolution-Recurrent Neural Network for Atrial Fibrillation Detection | 2019 | |
| Neural Networks | MINA | pyhealth.models.ecg.mina | MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals | 2019 | |

Examples of running ML and DL models can be found below, or directly at \examples\learning_examples\

**(iii) Evaluation Metrics** :

| Type | Abbr | Metric | Method |
|---|---|---|---|
| Binary Classification | average_precision_score | Compute micro/macro average precision (AP) from prediction scores | pyhealth.evaluation.xxx.get_avg_results |
| Binary Classification | roc_auc_score | Compute micro/macro ROC AUC score from prediction scores | pyhealth.evaluation.xxx.get_avg_results |
| Binary Classification | recall, precision, f1 | Get recall, precision, and f1 values | pyhealth.evaluation.xxx.get_predict_results |
| Multi Classification | To be done here | | |

**(iv) Supported Tasks**:

| Type | Abbr | Description | Method |
|---|---|---|---|
| Multi-classification | phenotyping | Predict the diagnosis code of a patient based on other information, e.g., procedures | \examples\data_generation\generate_phenotyping_xxx.py |
| Binary Classification | mortality prediction | Predict whether a patient may pass away during the hospital | \examples\data_generation\generate_mortality_xxx.py |
| Regression | ICU stay length pred | Forecast the length of an ICU stay | \examples\data_generation\generate_icu_length_xxx.py |

CHAPTER

# TWO

# ALGORITHM BENCHMARK

**The comparison among of implemented models** will be made available later with a benchmark paper. TBA soon :)

## 2.1 Installation

It is recommended to use **pip** for installation. Please make sure **the latest version** is installed, as PyHealth is updated frequently:

```
pip install pyhealth            # normal install
pip install --upgrade pyhealth  # or update if needed
pip install --pre pyhealth      # or include pre-release version for new features
```

Alternatively, you could clone and run setup.py file:

```
git clone https://github.com/yzhao062/pyhealth.git
cd pyhealth
pip install .
```

**Required Dependencies**:

- Python 3.5, 3.6, or 3.7
- combo>=0.0.8
- joblib
- numpy>=1.13
- numba>=0.35
- pandas>=0.25
- scipy>=0.20
- scikit_learn>=0.20
- tqdm
- torch (this should be installed manually)
- xgboost (this should be installed manually)
- xlrd >= 1.0.0

**Warning 1**: PyHealth has multiple neural network based models, e.g., LSTM, which are implemented in PyTorch. However, PyHealth does **NOT** install these DL libraries for you. This reduces the risk of interfering with your local copies. If you want to use neural-net based models, please make sure PyTorch is installed. Similarly, models depending on **xgboost**, would **NOT** enforce xgboost installation by default.

---

## 2.2 Examples

### 2.2.1 Quick Start for Data Processing

We propose the idea of standard template, a formalized schema for healthcare datasets. Ideally, as long as the data is scanned as the template we defined, the downstream task processing and the use of ML models will be easy and standard. In short, it has the following structure: **add a figure here**. The dataloader for different datasets can be found in examples/data_generation. Using "examples/data_generation/dataloader_mimic_demo.py" as an exmaple:

1. First read in patient, admission, and event tables.

```
from pyhealth.utils.utility import read_csv_to_df
patient_df = read_csv_to_df(os.path.join('data', 'mimic-iii-clinical-database-demo-
↪1.4', 'PATIENTS.csv'))
admission_df = read_csv_to_df(os.path.join('data', 'mimic-iii-clinical-database-
↪demo-1.4', 'ADMISSIONS.csv'))
...
```

2. Then invoke the parallel program to parse the tables in n_jobs cores.

```
from pyhealth.data.base_mimic import parallel_parse_tables
all_results = Parallel(n_jobs=n_jobs, max_nbytes=None, verbose=True)(
delayed(parallel_parse_tables)(
     patient_df=patient_df,
     admission_df=admission_df,
     icu_df=icu_df,
     event_df=event_df,
     event_mapping_df=event_mapping_df,
     duration=duration,
     save_dir=save_dir)
 for i in range(n_jobs))
```

3. The processed sequential data will be saved in the prespecified directory.

```
with open(patient_data_loc, 'w') as outfile:
    json.dump(patient_data_list, outfile)
```

The provided examples in PyHealth mainly focus on scanning the data tables in the schema we have, and **generate episode datasets**. For instance, "examples/data_generation/dataloader_mimic_demo.py" demonstrates the basic procedure of processing MIMIC III demo datasets.

1. The next step is to generate episode/sequence data for mortality prediction. See "examples/data_generation/generate_mortality_prediction_mimic_demo.py"

```
with open(patient_data_loc, 'w') as outfile:
    json.dump(patient_data_list, outfile)
```

By this step, the dataset has been processed for generating X, y for phenotyping prediction. **It is noted that the API across most datasets are similar**. One may easily replicate this procedure by calling the data generation scripts in \examples\data_generation. You may also modify the parameters in the scripts to generate the customized datasets.

**Preprocessed datasets are also available at \datasets\cms and \datasets\mimic**.

---

### 2.2.2 Quick Start for Running Predictive Models

**Note**: Before running examples, you need the datasets. Please download from the GitHub repository "datasets". You can either unzip them manually or running our script "00_extract_data_run_before_learning.py"

**Note**: "examples/learning_models/example_sequence_gpu_mortality.py" demonstrates the basic API of using GRU for mortality prediction. **It is noted that the API across all other algorithms are consistent/similar**.

**Note**: **If you do not have the preprocessed datasets yet, download the \datasets folder (cms.zip and mimic.zip) from PyHealth repository, and run \examples\learning_models\extract_data_run_before_learning.py to prepare/unzip the datasets.**

**Note**: For "certain examples", pretrained bert models are needed. You will need to download these pretrained models at:

- BERT+BioBERT: https://github.com/EmilyAlsentzer/clinicalBERT
- CharacterBERT+BioCharacterBERT: https://github.com/helboukkouri/character-bert

Please download, unzip, and save to ./auxiliary folder.

1. Setup the datasets. X and y should be in x_data and y_data, respectively.

```
# load pre-processed CMS dataset
from pyhealth.data.expdata_generator import sequencedata as expdata_generator

expdata_id = '2020.0810.data.mortality.mimic'
cur_dataset = expdata_generator(exp_id=exp_id)
cur_dataset.get_exp_data(sel_task='mortality', )
cur_dataset.load_exp_data()
```

2. Initialize a LSTM model, you may set up the parameters of the LSTM, e.g., n_epoch, learning_rate, etc,.

```
# initialize the model for training
from pyhealth.models.sequence.lstm import LSTM
# enable GPU
clf = LSTM(expmodel_id=expmodel_id, n_batchsize=20, use_gpu=True,
    n_epoch=100, gpu_ids='0,1')
clf.fit(cur_dataset.train, cur_dataset.valid)
```

3. Load the best shot of the training, predict on the test datasets

```
# load the best model for inference
clf.load_model()
clf.inference(cur_dataset.test)
pred_results = clf.get_results()
```

4. Evaluation on the model. Multiple metrics are supported.

```
# evaluate the model
from pyhealth.evaluation.evaluator import func
r = func(pred_results['hat_y'], pred_results['y'])
print(r)
```

## 2.3 API CheatSheet

Full API Reference: (https://pyhealth.readthedocs.io/en/latest/pyhealth.html). API cheatsheet for most learning models:

- `pyhealth.models.sequence._dlbase.fit()` : Fit a learning model.
- `pyhealth.models.sequence._dlbase.inference()` : Predict on X using the fitted estimator.
- **evaluator(y, y^hat)**: Model evaluation.

Model load and reload:

- `pyhealth.models.sequence._dlbase.load_model()` : Load the best model so far.

---

## 2.4 All Models

### 2.4.1 pyhealth.data package

#### Submodules

#### pyhealth.data.base module

**class** pyhealth.data.base.**Standard_Template**(*patient_id*)

Bases: `object`

Abstract class which can be inherited by various datasets, Key information and memory friendly information will be saved in the data dictionary. Otherwise, save the event and sequence location instead.

**abstract parse_admission**(*pd_df*, *mapping_dict=None*)

**parse_icu**(*pd_df*, *mapping_dict=None*)

**abstract parse_patient**(*pd_series*, *mapping_dict=None*)

#### pyhealth.data.base_cms module

Base class for CMS dataset

**class** pyhealth.data.base_cms.**CMS_Data**(*patient_id*, *procudure_cols*, *diagnosis_cols*)

Bases: *Standard_Template*

The data template to store CMS data. Customized fields can be added in each parse_xxx methods.

**Parameters**

**patient_id**
[str] Unique identifier for a patient.

**generate_phenotyping**(*pd_df*, *diagnosis_mapping_df*, *diagnosis_codes*, *diagnosis_dict*)

**parse_admission**(*pd_df*, *mapping_dict=None*)

**parse_event**(*pd_df*, *event_mapping_df*, *procedure_codes*, *procedure_dict*, *save_dir=''*)

**parse_patient**(*pd_series*)

### pyhealth.data.base_dataset module

**class** pyhealth.data.base_dataset.**BaseDataset**(*opt*)

Bases: Dataset, ABC

**static modify_commandline_options**(*parser*, *is_train*)

### pyhealth.data.base_mimic module

Base class for MIMIC dataset

**class** pyhealth.data.base_mimic.**MIMIC_Data**(*patient_id*, *time_duration*, *selection_method*)

Bases: *Standard_Template*

The data template to store MIMIC data. Customized fields can be added in each parse_xxx methods.

**Parameters**
patient_id

time_duration

selection_method

**generate_episode**(*pd_df*, *duration*, *event_mapping_df*, *var_list*)

**generate_episode_headers**(*var_list*)

Generate the header for episode file

**Parameters**
**var_list** –

**parse_admission**(*pd_df*)

**parse_event**(*pd_df*, *save_dir=''*, *event_mapping_df=''*, *var_list=None*)

**parse_icu**(*pd_df*, *mapping_dict=None*)

**parse_patient**(*pd_series*, *mapping_dict=None*)

**write_record**(*temp_list*, *temp_df*, *var*)

pyhealth.data.base_mimic.**parallel_parse_tables**(*patient_id_list*, *patient_df*, *admission_df*, *icu_df*, *event_df*, *event_mapping_df*, *duration*, *selection_method*, *var_list*, *save_dir*)

Parallel methods to process patient information in batches

**Parameters**

- **patient_id_list** –
- **patient_df** –
- **admission_df** –
- **icu_df** –
- **event_df** –
- **var_list** –

### pyhealth.data.expdata_generator module

class pyhealth.data.expdata_generator.**ecgdata**(*expdata_id*, *root_dir='.'*)

Bases: object

**get_exp_data**(*sel_task='diagnose'*, *shuffle=True*, *split_ratio=[0.64, 0.16, 0.2]*, *data_root=''*, *n_limit=-1*)

Parameters

---

**task**
[str, optional (default='phenotyping')] name of current healthcare task

**shuffle**
[bool, optional (default=True)] determine whether shuffle data or not

**split_ratio**
[list, optional (default=[0.64,0.16,0.2])] used for split whole data into train/valid/test

**data_root**
[str, optional (default='')] if data_root=='', use data in ./datasets; else use data in data_root

**n_limit**
[int, optional (default = -1)] used for sample N-data not for all data, if n_limit==-1, use all data

**load_exp_data**()

**show_data**(*k=3*)

Parameters

class pyhealth.data.expdata_generator.**imagedata**(*expdata_id*, *root_dir='.'*)

Bases: object

**get_exp_data**(*sel_task='diagnose'*, *shuffle=True*, *split_ratio=[0.64, 0.16, 0.2]*, *data_root=''*, *n_limit=-1*)

Parameters

---

**task**
[str, optional (default='phenotyping')] name of current healthcare task

**shuffle**
[bool, optional (default=True)] determine whether shuffle data or not

**split_ratio**
[list, optional (default=[0.64,0.16,0.2])] used for split whole data into train/valid/test

**data_root**
[str, (default='')] use data in data_root

**n_limit**
[int, optional (default = -1)] used for sample N-data not for all data, if n_limit==-1, use all data

load_exp_data()

show_data(*k=3*)
Parameters

class pyhealth.data.expdata_generator.**sequencedata**(*expdata_id*, *root_dir='.'*)
Bases: object

get_exp_data(*sel_task='phenotyping'*, *shuffle=True*, *split_ratio=[0.64, 0.16, 0.2]*, *data_root=''*, *n_limit=-1*)
Parameters

---

**task**
[str, optional (default='phenotyping')] name of current healthcare task

**shuffle**
[bool, optional (default=True)] determine whether shuffle data or not

**split_ratio**
[list, optional (default=[0.64,0.16,0.2])] used for split whole data into train/valid/test

**data_root**
[str, optional (default='')] if data_root==", use data in ./datasets; else use data in data_root

**n_limit**
[int, optional (default = -1)] used for sample N-data not for all data, if n_limit==-1, use all data

load_exp_data()

show_data(*k=3*)
Parameters

class pyhealth.data.expdata_generator.**textdata**(*expdata_id*, *root_dir='.'*)
Bases: object

get_exp_data(*sel_task='diagnose'*, *shuffle=True*, *split_ratio=[0.64, 0.16, 0.2]*, *data_root=''*, *n_limit=-1*)
Parameters

---

**task**
[str, optional (default='phenotyping')] name of current healthcare task

**shuffle**
[bool, optional (default=True)] determine whether shuffle data or not

**split_ratio**
[list, optional (default=[0.64,0.16,0.2])] used for split whole data into train/valid/test

**data_root**
[str, (default='')] use data in data_root

**n_limit**
[int, optional (default = -1)] used for sample N-data not for all data, if n_limit==-1, use all data

load_exp_data()

show_data(*k=3*)

Parameters

#### pyhealth.data.mimic_clean_methods module

MIMIC dataset handling. Adapted and modified from https://github.com/YerevaNN/mimic3-benchmarks/blob/master/mimic3benchmark/preprocessing.py

pyhealth.data.mimic_clean_methods.**clean_crr**(*df*)

pyhealth.data.mimic_clean_methods.**clean_dbp**(*df*)

pyhealth.data.mimic_clean_methods.**clean_fio2**(*df*)

pyhealth.data.mimic_clean_methods.**clean_height**(*df*)

pyhealth.data.mimic_clean_methods.**clean_lab**(*df*)

pyhealth.data.mimic_clean_methods.**clean_o2sat**(*df*)

pyhealth.data.mimic_clean_methods.**clean_sbp**(*df*)

pyhealth.data.mimic_clean_methods.**clean_temperature**(*df*)

pyhealth.data.mimic_clean_methods.**clean_weight**(*df*)

#### pyhealth.data.rnn_reader module

**class** pyhealth.data.rnn_reader.**DatasetReader**(*data*)

Bases: *BaseDataset*

pyhealth.data.rnn_reader.**time_series_get**(*fpath*)

#### Module contents

### 2.4.2 pyhealth.evaluation package

#### Submodules

#### pyhealth.evaluation.binaryclass module

pyhealth.evaluation.binaryclass.**evaluator**(*hat_y*, *y*)

pyhealth.evaluation.binaryclass.**get_avg_results**(*hat_y*, *y*)

pyhealth.evaluation.binaryclass.**get_predict_results**(*hat_y*, *y*)

#### pyhealth.evaluation.evaluator module

pyhealth.evaluation.evaluator.**check_evalu_type**(*hat_y*, *y*)

pyhealth.evaluation.evaluator.**func**(*hat_y*, *y*, *evalu_type=None*)

#### pyhealth.evaluation.mortality module

#### pyhealth.evaluation.multilabel module

pyhealth.evaluation.multilabel.**evaluator**(*hat_y*, *y*)

pyhealth.evaluation.multilabel.**get_avg_results**(*hat_y*, *y*)

pyhealth.evaluation.multilabel.**get_top_k_results**(*hat_y*, *y*, *k=1*)

#### pyhealth.evaluation.phenotyping module

#### Module contents

### 2.4.3 pyhealth.models.sequence package

#### Submodules

#### pyhealth.models.sequence.dipole module

**class** pyhealth.models.sequence.dipole.**ConcatenationAttention**(*hidden_size*, *attention_dim=16*, *device=None*)

Bases: Module

**forward**(*input_data*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

**class** pyhealth.models.sequence.dipole.**Dipole**(*expmodel_id='test.new'*, *n_epoch=100*, *n_batchsize=5*, *learn_ratio=0.0001*, *weight_decay=0.0001*, *n_epoch_saved=1*, *attention_type='location_based'*, *attention_dim=8*, *embed_size=16*, *hidden_size=8*, *output_size=8*, *bias=True*, *dropout=0.5*, *batch_first=True*, *loss_name='L1LossSigmoid'*, *target_repl=False*, *target_repl_coef=0.0*, *aggregate='sum'*, *optimizer_name='adam'*, *use_gpu=False*, *gpu_ids='0'*)

Bases: BaseControler

**fit**(*train_data*, *valid_data*, *assign_task_type=None*)

Parameters

---

**train_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**

predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

**load_model**(*loaded_epoch=''*, *config_file_path=''*, *model_file_path=''*)

Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

class pyhealth.models.sequence.dipole.**GeneralAttention**(*hidden_size*, *device*)

Bases: Module

**forward**(*input_data*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

class pyhealth.models.sequence.dipole.**LocationAttention**(*hidden_size*, *device*)

Bases: Module

**forward**(*input_data*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

class pyhealth.models.sequence.dipole.**callPredictor**(*input_size=None*, *embed_size=16*, *hidden_size=8*, *output_size=10*, *bias=True*, *dropout=0.5*, *batch_first=True*, *label_size=1*, *attention_type='location_based'*, *attention_dim=8*, *device=None*)

Bases: Module

**forward**(*input_data*)

Parameters

'M': shape (batchsize, n_timestep) 'cur_M': shape (batchsize, n_timestep) 'T': shape (batchsize, n_timestep)

}

Return

---

all_output, shape (batchsize, n_timestep, n_labels)

predict output of each time step

cur_output, shape (batchsize, n_labels)

predict output of last time step

**training:** bool

### pyhealth.models.sequence.embedgru module

class pyhealth.models.sequence.embedgru.**EmbedGRU**(*expmodel_id='test.new'*, *n_epoch=100*, *n_batchsize=5*, *learn_ratio=0.0001*, *weight_decay=0.0001*, *n_epoch_saved=1*, *embed_size=16*, *layer_hidden_sizes=[10, 20, 15]*, *bias=True*, *dropout=0.5*, *bidirectional=True*, *batch_first=True*, *loss_name='L1LossSigmoid'*, *target_repl=False*, *target_repl_coef=0.0*, *aggregate='sum'*, *optimizer_name='adam'*, *use_gpu=False*, *gpu_ids='0'*)

Bases: BaseControler

**fit**(*train_data*, *valid_data*, *assign_task_type=None*)

Parameters

---

**train_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**

predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

**load_model**(*loaded_epoch=''*, *config_file_path=''*, *model_file_path=''*)

Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

class pyhealth.models.sequence.embedgru.**callPredictor**(*input_size=None*, *embed_size=16*, *layer_hidden_sizes=[10, 20, 15]*, *num_layers=3*, *bias=True*, *dropout=0.5*, *bidirectional=True*, *batch_first=True*, *label_size=1*)

Bases: Module

**forward**(*input_data*)

Parameters

'M': shape (batchsize, n_timestep) 'cur_M': shape (batchsize, n_timestep) 'T': shape (batchsize, n_timestep)

}

Return

---

all_output, shape (batchsize, n_timestep, n_labels)

predict output of each time step

cur_output, shape (batchsize, n_labels)

predict output of last time step

**training:** bool

### pyhealth.models.sequence.gru module

**class** pyhealth.models.sequence.gru.**GRU**(*expmodel_id='test.new'*, *n_epoch=100*, *n_batchsize=5*, *learn_ratio=0.0001*, *weight_decay=0.0001*, *n_epoch_saved=1*, *layer_hidden_sizes=[10, 20, 15]*, *bias=True*, *dropout=0.5*, *bidirectional=True*, *batch_first=True*, *loss_name='L1LossSigmoid'*, *target_repl=False*, *target_repl_coef=0.0*, *aggregate='sum'*, *optimizer_name='adam'*, *use_gpu=False*, *gpu_ids='0'*)

Bases: BaseControler

**fit**(*train_data*, *valid_data*, *assign_task_type=None*)

Parameters

---

**train_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**

predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

**load_model**(*loaded_epoch=''*, *config_file_path=''*, *model_file_path=''*)

Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

class pyhealth.models.sequence.gru.**callPredictor**(*input_size=None*, *layer_hidden_sizes=[10, 20, 15]*, *num_layers=3*, *bias=True*, *dropout=0.5*, *bidirectional=True*, *batch_first=True*, *label_size=1*)

Bases: Module

**forward**(*input_data*)

Parameters

'M': shape (batchsize, n_timestep) 'cur_M': shape (batchsize, n_timestep) 'T': shape (batchsize, n_timestep)

}

Return

---

all_output, shape (batchsize, n_timestep, n_labels)

predict output of each time step

cur_output, shape (batchsize, n_labels)

predict output of last time step

**training:** bool

### pyhealth.models.sequence.lstm module

class pyhealth.models.sequence.lstm.**LSTM**(*expmodel_id='test.new'*, *n_epoch=100*, *n_batchsize=5*, *learn_ratio=0.0001*, *weight_decay=0.0001*, *n_epoch_saved=1*, *layer_hidden_sizes=[10, 20, 15]*, *bias=True*, *dropout=0.5*, *bidirectional=True*, *batch_first=True*, *loss_name='L1LossSigmoid'*, *target_repl=False*, *target_repl_coef=0.0*, *aggregate='sum'*, *optimizer_name='adam'*, *use_gpu=False*, *gpu_ids='0'*)

Bases: BaseControler

**fit**(*train_data*, *valid_data*, *assign_task_type=None*)

Parameters

---

**train_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**
predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

`load_model`(*loaded_epoch=''*, *config_file_path=''*, *model_file_path=''*)
Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

`class` pyhealth.models.sequence.lstm.`callPredictor`(*input_size=None*, *layer_hidden_sizes=[10, 20, 15]*, *num_layers=3*, *bias=True*, *dropout=0.5*, *bidirectional=True*, *batch_first=True*, *label_size=1*)

Bases: `Module`

`forward`(*input_data*)
Parameters

'M': shape (batchsize, n_timestep) 'cur_M': shape (batchsize, n_timestep) 'T': shape (batchsize, n_timestep)

}

Return

---

all_output, shape (batchsize, n_timestep, n_labels)

predict output of each time step

cur_output, shape (batchsize, n_labels)

predict output of last time step

`training:` `bool`

### pyhealth.models.sequence.raim module

class pyhealth.models.sequence.raim.**RAIM**(*expmodel_id='test.new'*, *n_epoch=100*, *n_batchsize=5*, *learn_ratio=0.0001*, *weight_decay=0.0001*, *n_epoch_saved=1*, *window_size=3*, *hidden_size=8*, *output_size=8*, *bias=True*, *dropout=0.5*, *batch_first=True*, *loss_name='L1LossSigmoid'*, *target_repl=False*, *target_repl_coef=0.0*, *aggregate='sum'*, *optimizer_name='adam'*, *use_gpu=False*, *gpu_ids='0'*)

Bases: BaseControler

Recurrent Attentive and Intensive Model(RAIM) for jointly analyzing continuous monitoring data and discrete clinical events

**fit**(*train_data*, *valid_data*, *assign_task_type=None*)

Parameters

---

**train_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**
predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

**load_model**(*loaded_epoch=''*, *config_file_path=''*, *model_file_path=''*)

Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

class pyhealth.models.sequence.raim.**RaimExtract**(*input_size*, *window_size*, *hidden_size*)

Bases: Module

**forward**(*input_data*, *h_t_1*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**training:** bool

class pyhealth.models.sequence.raim.**callPredictor**(*input_size=None*, *window_size=3*, *hidden_size=16*, *output_size=8*, *batch_first=True*, *dropout=0.5*, *label_size=1*, *device=None*)

Bases: Module

**forward**(*input_data*)

Parameters

'M': shape (batchsize, n_timestep) 'cur_M': shape (batchsize, n_timestep) 'T': shape (batchsize, n_timestep)

}

Return

---

all_output, shape (batchsize, n_timestep, n_labels)

predict output of each time step

cur_output, shape (batchsize, n_labels)

predict output of last time step

**training:** bool

### pyhealth.models.sequence.retain module

class pyhealth.models.sequence.retain.**Retain**(*expmodel_id='test.new'*, *n_epoch=100*, *n_batchsize=5*, *learn_ratio=0.0001*, *weight_decay=0.0001*, *n_epoch_saved=1*, *embed_size=16*, *hidden_size=8*, *bias=True*, *dropout=0.5*, *batch_first=True*, *loss_name='L1LossSigmoid'*, *target_repl=False*, *target_repl_coef=0.0*, *aggregate='sum'*, *optimizer_name='adam'*, *use_gpu=False*, *gpu_ids='0'*)

Bases: BaseControler

**fit**(*train_data*, *valid_data*, *assign_task_type=None*)

Parameters

---

**train_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**
predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

`load_model`(*loaded_epoch=''*)
Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

`class` pyhealth.models.sequence.retain.**RetainAttention**(*embed_size*, *hidden_size*, *device*)
Bases: `Module`

`forward`(*data_alpha*, *data_beta*, *data_embed*, *data_mask*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

`training: bool`

`class` pyhealth.models.sequence.retain.**callPredictor**(*input_size=None*, *embed_size=16*, *hidden_size=8*, *bias=True*, *dropout=0.5*, *batch_first=True*, *label_size=1*, *device=None*)
Bases: `Module`

**forward**(*input_data*)

Parameters

'M': shape (batchsize, n_timestep) 'cur_M': shape (batchsize, n_timestep) 'T': shape (batchsize, n_timestep)

}

Return

---

all_output, shape (batchsize, n_timestep, n_labels)

predict output of each time step

cur_output, shape (batchsize, n_labels)

predict output of last time step

**training:** bool

### pyhealth.models.sequence.rf module

**class** pyhealth.models.sequence.rf.**RandomForest**(*expmodel_id='test.new'*, *n_estimators=100*, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='auto'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*, *ccp_alpha=0.0*, *max_samples=None*)

Bases: object

**fit**(*data_dict*, *X=None*, *y=None*, *assign_task_type=None*)

Parameters

---

**train_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**

[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

Returns

---

self : object

Fitted estimator.

`get_results()`

**Load saved prediction results in current ExpID**
truth_value: proj_root/experiments_records/*****(exp_id)/results/y predict_value: proj_root/experiments_records/*****(exp_id)/results/hat_y xxx represents the loaded model

`inference(data_dict, X=None, y=None)`

Parameters

---

**test_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input test samples dict.

`load_model()`

Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

### pyhealth.models.sequence.stagenet module

StageNet model. Adapted and modified from

https://github.com/v1xerunt/StageNet

`class pyhealth.models.sequence.stagenet.StageNet(expmodel_id='test.new', n_epoch=100, n_batchsize=5, learn_ratio=0.0001, weight_decay=0.0001, n_epoch_saved=1, hidden_size=384, conv_size=10, levels=3, dropconnect=0.3, dropout=0.3, dropres=0.3, batch_first=True, loss_name='L1LossSigmoid', target_repl=False, target_repl_coef=0.0, aggregate='sum', optimizer_name='adam', use_gpu=False, gpu_ids='0')`

Bases: `BaseControler`

StageNet: Stage-Aware Neural Networks for Health Risk Prediction.

`fit(train_data, valid_data, assign_task_type=None)`

Parameters

---

**train_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**
predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

`load_model`(*loaded_epoch=''*, *config_file_path=''*, *model_file_path=''*)

Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

`class` pyhealth.models.sequence.stagenet.`callPredictor`(*input_dim=None*, *hidden_dim=384*, *conv_size=10*, *levels=3*, *dropconnect=0.3*, *dropout=0.3*, *dropres=0.3*, *label_size=None*, *device=None*)

Bases: `Module`

`cumax`(*x*, *mode='l2r'*)

`forward`(*input_data*)

Parameters

'M': shape (batchsize, n_timestep) 'cur_M': shape (batchsize, n_timestep) 'T': shape (batchsize, n_timestep)

}

Return

---

all_output, shape (batchsize, n_timestep, n_labels)

predict output of each time step

cur_output, shape (batchsize, n_labels)

predict output of last time step

**step**(*inputs*, *c_last*, *h_last*, *interval*)

**training:** bool

### pyhealth.models.sequence.tlstm module

**class** pyhealth.models.sequence.tlstm.**callPredictor**(*input_size=None*, *hidden_size=16*, *output_size=8*, *batch_first=True*, *dropout=0.5*, *label_size=1*, *device=None*)

Bases: Module

**forward**(*input_data*)

Parameters

'M': shape (batchsize, n_timestep) 'cur_M': shape (batchsize, n_timestep) 'T': shape (batchsize, n_timestep)

}

Return

---

all_output, shape (batchsize, n_timestep, n_labels)

predict output of each time step

cur_output, shape (batchsize, n_labels)

predict output of last time step

**training:** bool

**class** pyhealth.models.sequence.tlstm.**tLSTM**(*expmodel_id='test.new'*, *n_epoch=100*, *n_batchsize=5*, *learn_ratio=0.0001*, *weight_decay=0.0001*, *n_epoch_saved=1*, *hidden_size=8*, *output_size=8*, *bias=True*, *dropout=0.5*, *batch_first=True*, *loss_name='L1LossSigmoid'*, *target_repl=False*, *target_repl_coef=0.0*, *aggregate='sum'*, *optimizer_name='adam'*, *use_gpu=False*, *gpu_ids='0'*)

Bases: BaseControler

**Time-Aware LSTM (T-LSTM), A kind of time-aware RNN neural network;**
Used to handle irregular time intervals in longitudinal patient records.

**fit**(*train_data*, *valid_data*, *assign_task_type=None*)

Parameters

---

**train_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**
predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

**load_model**(*loaded_epoch=''*, *config_file_path=''*, *model_file_path=''*)

Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

**class** pyhealth.models.sequence.tlstm.**tLSTMCell**(*input_size*, *hidden_size*)

Bases: Module

**forward**(*data_x*, *data_t*, *h_t_1*, *c_t_1*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

---

**Note:** Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

**reset_parameters**()

**training:** bool

#### pyhealth.models.sequence.xgboost module

#### Module contents

### 2.4.4 pyhealth.models.image package

#### Submodules

#### pyhealth.models.image.typicalcnn module

class pyhealth.models.image.typicalcnn.**TypicalCNN**(*expmodel_id='test.new'*, *cnn_name='resnet18'*, *pretrained=False*, *n_epoch=100*, *n_batchsize=5*, *load_size=255*, *crop_size=224*, *learn_ratio=0.0001*, *weight_decay=0.0001*, *n_epoch_saved=1*, *bias=True*, *dropout=0.5*, *batch_first=True*, *loss_name='L1LossSoftmax'*, *aggregate='sum'*, *optimizer_name='adam'*, *use_gpu=False*, *gpu_ids='0'*)

Bases: BaseControler

Several typical & popular CNN networks for medical image prediction

Parameters

**cnn_name**
[str, optional (default = 'resnet18')] name of typical/popular CNN networks

**pretrained**
[bool, optional (default = False)] used for pre-trained model load, True -> load pretrained model; False -> not load

**n_epoch**
[int, optional (default = 100)] number of epochs with the initial learning rate

**n_batchsize**
[int, optional (default = 5)] batch size for model training

**load_size**
[int, optional (default = 255)] scale images to this size

**crop_size**
[int, optional (default = 224)] crop load_sized image into to this size

**learn_ratio**
[float, optional (default = 1e-4)] initial learning rate for adam

**weight_decay**
[float, optional (default = 1e-4)] weight decay (L2 penalty)

**n_epoch_saved**
[int, optional (default = 1)] frequency of saving checkpoints at the end of epochs

**bias**
[bool, optional (default = True)] If False, then the layer does not use bias weights b_ih and b_hh.

**dropout**
[float, optional (default = 0.5)] If non-zero, introduces a Dropout layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to dropout.

**batch_first**
[bool, optional (default = False)] If True, then the input and output tensors are provided as (batch, seq, feature).

**loss_name**
[str, optional (default='SigmoidCELoss')] Name or objective function.

**use_gpu**
[bool, optional (default=False)] If yes, use GPU resources; else use CPU resources

**gpu_ids**
[str, optional (default='')] If yes, assign concrete used gpu ids such as '0,2,6'; else use '0'

`fit`(*train_data*, *valid_data*, *assign_task_type=None*)

Parameters

---

**train_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input train samples dict.

**valid_data**
[{]

'x':list[episode_file_path], 'y':list[label], 'l':list[seq_len], 'feat_n': n of feature space, 'label_n': n of label space }

The input valid samples dict.

**assign_task_type: str (default = None)**
predifine task type to model mapping <feature, label> current support ['binary','multiclass','multilabel','regression']

Returns

---

self : object

Fitted estimator.

`load_model`(*loaded_epoch=''*, *config_file_path=''*, *model_file_path=''*)

Parameters

---

loaded_epoch : str, loaded model name

we save the model by <epoch_count>.epoch, latest.epoch, best.epoch

Returns

---

self : object

loaded estimator.

#### Module contents

### 2.4.5 pyhealth.utils package

#### Submodules

#### pyhealth.utils.check module

pyhealth.utils.check.**check_expdata_dir**(*expdata_id*)

**Check whether the exp data folder exist,**
If not, will create the folder

Parameters

**expdata_id**
[str, optional (default='init.test')] name of current experiment data

pyhealth.utils.check.**check_model_dir**(*expmodel_id*)

**Check whether the checkouts/results folders of current experiment(exp_id) exist,**
If not, will create both folders

Parameters

**expmodel_id**
[str, optional (default='init.test')] name of current experiment

pyhealth.utils.check.**label_check**(*y*, *hat_y=None*, *assign_task_type=None*)

#### pyhealth.utils.utility module

A set of utility functions to support outlier detection.

pyhealth.utils.utility.**check_parameter**(*param*, *low=-2147483647*, *high=2147483647*, *param_name=''*, *include_left=False*, *include_right=False*)

Check if an input is within the defined range.

**Parameters**

- **param** (*int, float*) – The input parameter to check.
- **low** (*int, float*) – The lower bound of the range.
- **high** (*int, float*) – The higher bound of the range.
- **param_name** (*str, optional (default='')*) – The name of the parameter.
- **include_left** (*bool, optional (default=False)*) – Whether includes the lower bound (lower bound <=).
- **include_right** (*bool, optional (default=False)*) – Whether includes the higher bound (<= higher bound).

**Returns**
**within_range** – Whether the parameter is within the range of (low, high)

**Return type**
bool or raise errors

pyhealth.utils.utility.**make_dirs_if_not_exists**(*save_dir*)

pyhealth.utils.utility.**read_csv_to_df**(*file_loc*, *header_lower=True*, *usecols=None*, *dtype=None*, *low_memory=True*, *encoding=None*)

Read in csv files with necessary processing

**Parameters**

- **file_loc** –
- **header_lower** –
- **low_memory** –

pyhealth.utils.utility.**read_excel_to_df**(*file_loc*, *header_lower=True*, *usecols=None*, *dtype=None*, *low_memory=True*, *encoding=None*)

Read in excel files with necessary processing

**Parameters**

- **file_loc** –
- **header_lower** –
- **low_memory** –

### pyhealth.utils.utility_parallel module

A set of utility functions to support parallel computation.

pyhealth.utils.utility_parallel.**partition_estimators**(*n_estimators*, *n_jobs*)

Private function used to partition estimators between jobs.

pyhealth.utils.utility_parallel.**tqdm_joblib**(*tqdm_object*)

Context manager to patch joblib to report into tqdm progress bar given as argument

pyhealth.utils.utility_parallel.**unfold_parallel**(*lists*, *n_jobs*)

Internal function to unfold the results returned from the parallization

**Parameters**

- **lists** (*list*) – The results from the parallelization operations.
- **n_jobs** (*optional (default=1)*) – The number of jobs to run in parallel for both *fit* and *predict*. If -1, then the number of jobs is set to the number of cores.

**Returns**

**result_list** – The list of unfolded result.

**Return type**

list

#### Module contents

## 2.5 About us

### 2.5.1 Core Development & Advisory Team

Yue Zhao (Ph.D. Student @ Carnegie Mellon University; initialized the project in Jun 2020): Homepage

Dr. Zhi Qiao (Associate ML Director @ IQVIA; initialized the project in Jun 2020): LinkedIn

Dr. Xiao Cao (Director, Analytics Center of Excellence of IQVIA @ IQVIA; initialized the project in Jun 2020)

Dr. Lucas Glass (Global Head, Analytics Center of Excellence @ IQVIA; initialized the project in Jun 2020): LinkedIn

Xiyang Hu (Ph.D. Student @ Carnegie Mellon University; initialized the project in Jun 2020): Homepage

Prof. Jimeng Sun (Professor @ University of Illinois Urbana-Champaign; initialized the project in Jun 2020): **`SUNLAB <<http://sunlab.org/>`_**

## 2.6 Frequently Asked Questions

---

### 2.6.1 Blueprint & Development Plan

The long term goal of PyHealth is to become a comprehensive healthcare AI toolkit that supports beyond EHR data, but also the images and clinical notes.

This is the central place to track important things to be fixed/added:

- The support of image datasets and clinical notes
- The compatibility and the support of OMOP format datasets
- Model persistence (save, load, and portability)
- The release of a benchmark paper with PyHealth
- Add contact channel with Gitter
- Support additional languages, see Manage Translations

Feel free to open on issue report if needed. See Issues.

### 2.6.2 Inclusion Criteria

Similarly to Similarly to scikit-learn, We mainly consider well-established algorithms for inclusion. A rule of thumb is at least two years since publication, 50+ citations, and usefulness.

However, we encourage the author(s) of newly proposed models to share and add your implementation into combo for boosting ML accessibility and reproducibility. This exception only applies if you could commit to the maintenance of your model for at least two year period.

---

### References

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# BIBLIOGRAPHY

[ABre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

# PYTHON MODULE INDEX

## p

# INDEX

## G



## I



## L

## M



## P

## R



## S



## T



## U



## W