PyTrial

Release 0.0.1

Zifeng Wang, Brandon Theodorou, Tianfan Fu, Jingtang Ma

Jun 11, 2023

GETTING STARTED

PyTrial is an easy-to-use **Python package** for a series of AI for drug development tasks. **Clinical trial** is the major step of the drug development process, where phase I, II, III, & IV trials are performed to comprehensively evaluate the efficacy and safety of a new drug.

PyTrial is featured for the following tasks and we are adding more!

- **Patient outcome prediction**: predict the patient outcomes using tabular or sequential patient data.
- **Trial site selection**: pick the best trial sites considering multiple objectives.
- **Trial outcome prediction**: predict the trial outcomes using tabular or sequential trial data.
- **Patient-trial matching**: match trials' elibility criteria to patients' EHRs for participant recruitment of clinical trials.
- **Trial similarity search**: encode and retrieve similar clinical trials based on trial design documents.
- **Trial data simulation**: generate synthetic EHRs or trial patient records in table or sequence.

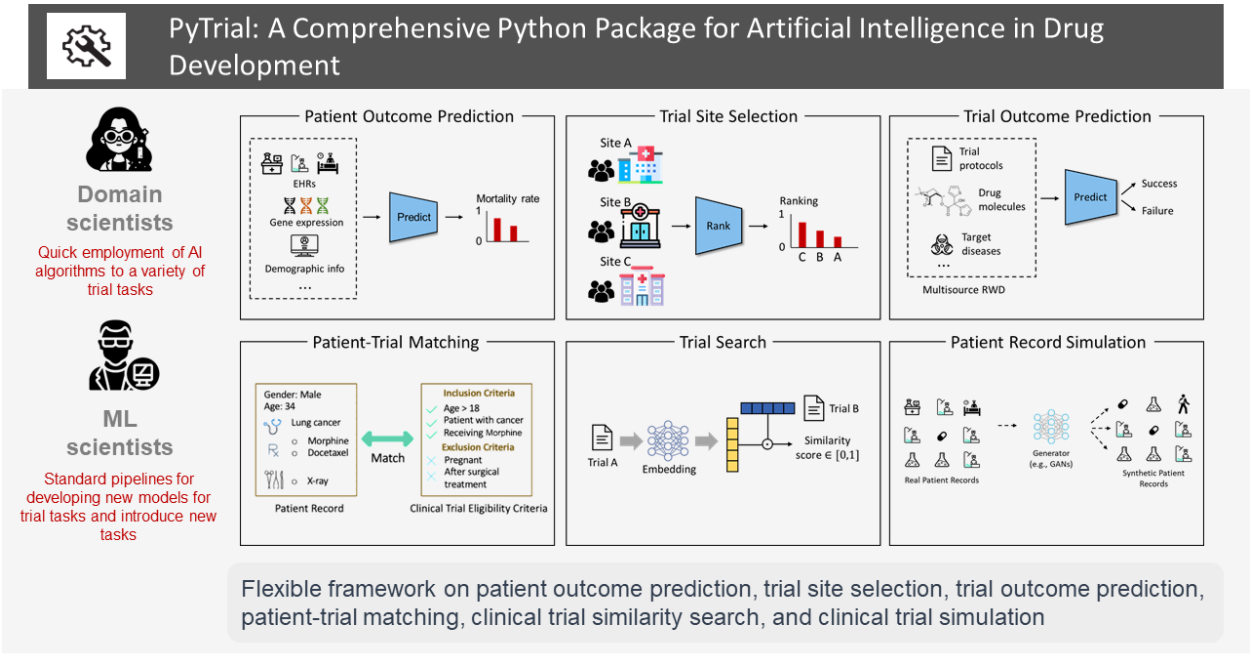

Fig. 1: The demonstration of clinical trial tasks supported by PyTrial.

If you find this package useful, please consider citing it in your scientific publications:

```
@misc{pytrial2022,
    title = {PyTrial: A Python Package for Artificial Intelligence in Drug Development},
    author = {Wang, Zifeng and Theodorou, Brandon and Fu, Tianfan and Sun, Jimeng},
    year = {2022},
    month = {11},
    organization = {SunLab@UIUC},
    note = {Version 0.0.1},
}
```

CHAPTER

ONE

INSTALLATION

transtab was tested on Python 3.7+, PyTorch 1.8.0+. Please follow the Installation instructions below for the torch version and CUDA device you are using:

PyTorch Installation Instructions.

After that, **PyTrial** can be downloaded directly using **pip**.

```
pip install pytrial
```

or

```
pip install git+https://github.com/RyanWangZf/pytrial.git
```

Alternatively, you can clone the project and install from local

```
git clone https://github.com/RyanWangZf/pytrial.git
cd pytrial
pip install .
```

Troubleshooting:

1. TBD

CHAPTER

TWO

TUTORIAL

We provide the following tutorials to help users get started with our **PyTrial**. After go through all these chapters, you will become the expert in AI for clinical trials and are ready to explore the frontier of this field. We are more than happy to welcome you to join us to revolutionize the drug development process via the cutting-edge AI technologies!

2.1 Overview of PyTrial

Table of Contents

As the first chapter of the tutorial, we want to deliver the basic idea of how we design **PyTrial** in order to help deploy artificial intelligence (AI) algorithms in the real-world drug development process.

2.1.1 Principle: Input & Output Define the Task

We believe the top priority is to ensure that PyTrial is **easy-to-use**. We are committed to developing a pipeline for each task with *a couple lines of codes*, for example,

```
# import the model
from pytrial.tasks.indiv_outcome.tabular import transtab

# initialize the model
model = transtab()

# fit the model
model.fit(train_data, valid_data)

# predict the outcome
model.predict(test_data)

# save the model to disk
model.save_model('./checkpoints')
```

Without vagueness, this pipeline has an intuitive logic for all machine learning people. We try to keep this interface as simple as possible and its style consistent across **all tasks and models**. That is to say, all models should have the same common functions like `fit`, `predict`, `save_model`, `load_model`, etc. In this way, users can easily switch between different tasks and models without any extra effort to study the interface.

The only thing that remains to care about is then the `input` and `output` of the model, which defines the task.

2.1.2 Code Hierarchy

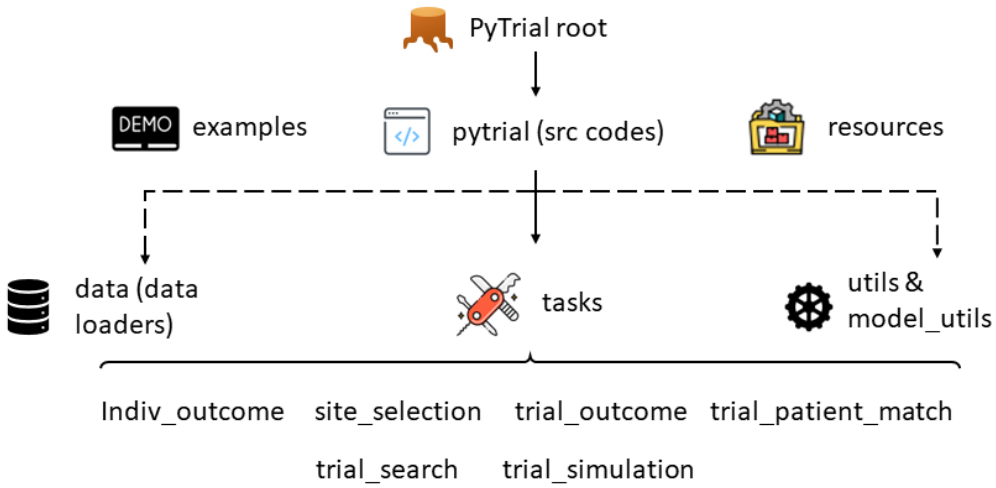

Fig. 1: The hierarchy of PyTrial source codes, corresponds to the one in https://github.com/RyanWangZf/PyTrial.

The hierarchy of the PyTrial repository is illustrated by the above figure. On the first layer, we have the `pytrial` folder, which contains all the source codes of PyTrial. The other two are the `examples` that store the jupyter notebook examples for each model and the `resources` folder that stores some tools and preprocessed data.

In most scenarios, users only need to use the `pytrial` folder. The `examples` folder is for users who want to learn how to use PyTrial. The `resources` folder is for developers who want to add new models or tasks to PyTrial.

Specifically, the `pytrial` folder contains the following subfolders:

- *indiv_outcome*: contains the source codes for individualized outcome prediction tasks.
- *site_selection*: contains the source codes for site selection tasks.
- *trial_outcome*: contains the source codes for trial outcome prediction tasks.
- *trial_patient_match*: contains the source codes for trial patient matching tasks.
- *trial_search*: contains the source codes for trial search tasks.
- *trial_simulation*: contains the source codes for trial patient record simulation tasks.

We will play with the models under each task folder, e.g., `trial_search.trial2vec`. In some cases, the task has its specific input format, for instance, the patient-trial matching task requires input patient and trial data in the form of `trial_patient_match.data.PatientData` and `trial_patient_match.data.TrialData`, respectively.

Don't worry, we will go through these tasks one by one with concrete examples!

2.2 PyTrial API & Pipeline

Table of Contents

As described in *Intro 1: Overview of PyTrial*, **PyTrial** maintains a consistent user interface, including common functions like `fit`, `predict`, `save_model`, and `load_model`. Each task is defined by its input and output data.

Therefore, the general pipeline is as follows, we take a patient-level outcome prediction task as an example:

1. Prepare the input data for the task.

```
from pytrial.data.patient_data import TabularPatientBase
from pytrial.utils.tabular_utils import MinMaxScaler
from pytrial.utils.tabular_utils import read_csv_to_df

# Read the data
df = read_csv_to_df('./tabular_patient_outcome_data.csv', index_col=0)
label = df['target_label']
df = df.drop(['target_label'], axis=1)

# Build the dataset for the specific task
# Here, we are working on individual patient level outcome prediction taking tabular␣
↪inputs.
dataset = TabularPatientBase(df,
    metadata={
        'transformers':
            {'age': MinMaxScaler()}, # specify the data transformation for specific␣
↪columns
    })
```

2. Import the models from the corresponding `pytrial.tasks` module.

```
from pytrial.tasks.indiv_outcome.tabular import LogisticRegression

model = LogisticRegression()
```

3. Train the model using the `fit` function.

```
model.fit(
    {
    'x': dataset,
    'y': label,
    }
)
```

4. Make the prediction using the `predict` function. And save the model using the `save_model` function.

```
# make predictions
ypred = model.predict({'x': dataset})
```

(continues on next page)

(continued from previous page)

```
# save the model
model.save_model('./model')
```

We can see that, except for the first step for data preparation, the rest of the steps are rather straightforward. For the sake of supporting the data preparation, we provide a set of basic dataset classes in the `pytrial.data` module. We also provide a set of children classes of them for the specific tasks, e.g., `pytrial.tasks.trial_patient_match.data.PatientData` and `pytrial.tasks.trial_patient_match.data.TrialData` considering the trial-patient matching task.

We will go through each task with concrete examples in the next chapters.

2.3 Basic Patient Data Class

Table of Contents

PyTrial offers several basic data classes for organizing patient and trial data:

- *Document: pytrial.data.patient_data*
- *Document: pytrial.data.trial_data*

They define the basic structure of the input data and then the convenience for the next model training and predicting. Thus, we use these standard data classes as the inputs for many tasks. Here, we will show two examples for patient data building.

We categorize the input data into two types: **tabular** and **sequential** patient data. The former is the data that can be represented as a table: each row is a patient and each column is a feature. The latter is the data that each patient has a sequence of visits, where each visit has multiple features, e.g., events, lab tests, etc.

2.3.1 Patient Data: Tabular

Colab example is available at Example: data.patient_data.TabularPatientBase.

We have *pytrial.data.patient_data.TabularPatientBase* for the tabular patient data.

```
from pytrial.data.patient_data import TabularPatientBase
```

Consider we get patient data in `pandas.DataFrame` format but the raw features are a mixture of texts, numbers, and missing values. We usually need to preprocess the data before passing it to models. `TabularPatientBase` provides a convenient way to do this.

Let's first load the raw demo data for creating a `TabularPatientBase` instance.

```
# load the raw demo data
from pytrial.data.demo_data import load_trial_patient_tabular
data = load_trial_patient_tabular()
```

(continues on next page)

(continued from previous page)

```
# parse the raw data
df = data['data']
metadata = data['metadata']
```

Then, we can pass the raw dataframe to the target data class.

```
import rdt

# create a TabularPatientBase instance
patient_data = TabularPatientBase(
    df=df, # this contains the raw dataframe
    metadata= {

        'sdtypes': {
            'race': 'categorical',
            'target_label': 'boolean',
            }, # this contains the data types of each column

        'tranformers': {
            'race': rdt.transformers.FrequencyEncoder(),
            }, # this contains the transformers for each column
        },
    )
```

A list of available data transformers can be found on https://docs.sdv.dev/rdt/transformers-glossary/browse-transformers. By default, `TabularPatientBase` will *automatically* detect the data types of each column and apply the corresponding transformers, e.g., `rdt.transformers.FrequencyEncoder` for categorical features, which means you can leave the `metadata` empty all the time, like this:

```
# leave the metadata empty and let the class automatically detect the data types and␣
↪apply transformers
patient_data = TabularPatientBase(df=df)
```

However, sometimes you may want to customize the data types and transformers in case the automatically detected ones are wrong. That is why in the above example we assign `'race':'categorical'` amd `'race': rdt.tranformers.FrequencyEncoder()`, which will push the dataclass to follow our custom settings.

Please notice that we are allowed to just pass `'sdtypes'` for one column without specifying the corresponding transformer, where the dataclass will pick the default transformer for the passed data type. as the `'target_label':'boolean'` in the above example.

We can check the transformed tabular data by

```
# the transformed values
patient_data.df
```

Besides, we can actually transform the data back to its original format by

```
# transform the data back to its original format
df_raw = patient_data.reverse_transform()
```

Or pass another dataframe to the dataclass to be transformed like

```
# pass another dataframe to the dataclass to be transformed
df_prime_transformed = patient_data.transform(df_prime)
```

2.3.2 Patient Data: Sequence

Colab example is available at Example: data.patient_data.SequencePatientBase.

We have *pytrial.data.patient_data.SequencePatientBase* for the sequential patient data.

```
from pytrial.data.patient_data import SequencePatientBase
```

Load the raw demo data to see how to create a `SequencePatientBase` instance.

```
from pytrial.data.demo_data import load_synthetic_ehr_sequence
data = load_synthetic_ehr_sequence()
data.keys()
'''
dict_keys(['visit', 'feature', 'order', 'n_num_feature', 'y', 'voc', 'cat_cardinalities'])
'''

# the raw visit data
data['visit'][0]
'''
[
    [[0, 1, 2, 3, 5, 7, 41, 313, 1], [0, 1, 82], [2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14,␣
↪15, 51, 19, 26]],
    [[0, 1, 10, 69], [1, 4], [0, 2, 3, 6, 7, 41, 12, 13, 14, 16, 52, 54, 22, 28]]
]
'''

# the order
data['order']
'''
['diag', 'prod', 'med']
'''

# the vocabulary
data['voc']
'''
{'diag': <promptehr.data.Voc at 0x7f150c615cd0>,
'prod': <promptehr.data.Voc at 0x7f14af789750>,
'med': <promptehr.data.Voc at 0x7f14af7b7310>}
'''
```

In the above example, `data['visit']` should be a list of patients, where each patient is a list of visits, where each visit is a list of events. That is, `data['visit'][0]` is the visits of the first patient, where `data['visit'][0][0]` is the first visit.

It should be noted that inside `data['visit'][0][0]` there are three lists, where each list contains several events of the same type, for example, diagnosed diseases represented by ICD codes.

The `data['order']` is the order of the events in each visit, which should be the same for all patients. In the above example, `data['order']` is `['diag', 'prod', 'med']`, which means the first list of `data['visit'][0][0]` is the diagnosed diseases, and so on.

The `data['voc']` contains the vocabularies for each event type. Each voc objective should have the same format as *pytrial.data.vocab_data.Vocab*.

Once we have the raw data, we can create a `SequencePatientBase` instance.

```
# create a ``SequencePatientBase`` instance
seqdata = SequencePatientBase(
    data={'v':data['visit'], 'y':data['y'], 'x':data['feature']},
    metadata={
        'visit':{
            'mode':'dense',
            'order': data['order'] # need to parse the ``order`` here
            },
        'label':{'mode':'tensor'},
        'voc':data['voc'], # need to parse the ``voc`` here
        'max_visit':20,
        }
    )
```

The parameter `data` contains the raw data including visits, label, and baseline features; the parameter `metadata` customize the output data format.

Then, we can check the transformed data by

```
from torch.utils.data import DataLoader
from pytrial.data.patient_data import SeqPatientCollator # we need a collation function␣
↪to process the input SequencePatient dataset

# let's see the outputs
collate_fn = SeqPatientCollator()
loader = DataLoader(seqdata, batch_size=2, collate_fn=collate_fn, num_workers=0)
loader = iter(loader)
batch = next(loader)
print(batch.keys())
'''
dict_keys(['v', 'x', 'y'])
'''

batch['v'].keys()
'''
dict_keys(['diag', 'prod', 'med'])
'''

batch['v']['diag'][0]
'''
[[0, 1, 2, 3, 5, 7, 41, 313, 1], [0, 1, 10, 69]]
'''
```

The dataloader returns the visits with keys corresponding to `data['order']`, i.e., `['diag', 'prod', 'med']` in the above example. `batch['v']['diag'][0]` is the diagnosis events for the first patient, where there are two visits.

2.4 Basic Trial Data Class

Table of Contents

PyTrial offers several basic data classes for organizing patient and trial data:

- *Document: pytrial.data.patient_data*
- *Document: pytrial.data.trial_data*

They define the basic structure of the input data and then the convenience for the next model training and predicting. Thus, we use these standard data classes as the inputs for many tasks. Here, we will show two examples for trial data building.

2.4.1 Trial Data: Trial Document Data

Colab example is available at Example: data.trial_data.TrialDatasetBase.

We have a trial document data class *pytrial.data.trial_data.TrialDatasetBase* for organizing the trial data.

The first step is also loading the raw data:

```
from pytrial.data.demo_data import load_trial_outcome_data

# load raw demo data, will download from the remote if not available in local
data = load_trial_outcome_data()

data.keys()
'''
dict_keys(['data'])
'''

data['data']
'''
A pd.DataFrame
'''
```

Building a trial document dataset is as easy as building a tabular data like

```
from pytrial.data.trial_data import TrialDatasetBase

# build a trial document dataset
trial_dataset = TrialDatasetBase(data=data['data'].iloc[:100])

trial_dataset.df['inclusion_criteria'][:2]
'''
0    [inclusion criteria:, -  age >= 18 years., -  ...
1    [inclusion criteria:, -  rare tumor, -  metast...
Name: inclusion_criteria, dtype: object
'''
```

(continues on next page)

(continued from previous page)

```
len(trial_dataset.df['inclusion_criteria'].iloc[0])
'''
12
'''
```

In the above example, `trial_dataset.df` is the transformed version of the raw input dataframe. Specifically, it splits raw crieteria into a list of single criterion as:

```
trial_dataset.df['inclusion_criteria'][0]
'''

['inclusion criteria:',
'-  age >= 18 years.',
'-  histological or cytological diagnosis of metastatic stage iv or locally advanced,',
'amenable to local therapy with curative intent.',
...
]
'''
```

We can further use a BERT model to encode each criterion to get their criterion-level embeddings by

```
# get the embeddings of inclusion and exclusion criteria sentences
# using a pretrained BERT model
# need GPU to run this
trial_dataset.get_ec_sentence_embedding()

trial_dataset.inc_ec_embedding.shape
'''
torch.Size([1986, 768])
'''

len(trial_dataset.inc_vocab)
'''
1986
'''

trial_dataset.inc_vocab.idx2word[2]
'''
'-  age >= 18 years.'
'''
```

After we call `get_ec_sentence_embedding`, the dataset will have new attributes for inclusion/exclusion criteria and their embeddings.

2.5 Individual Patient Outcome Prediction

Table of Contents

Making individual outcome predictions is the basic AI for healthcare task. We need to specify the target response to predict, e.g., mortality $y \in [0, 1]$, readmission $y \in [0, 1]$, length of stay $y \in [0, \infty]$, transform the input trial patient records to either static descriptive features $x \in \mathbb{R}^d$ where d is the number of features or sequential event features $x \in \mathbb{R}^{v \times d}$ where v is the number of visits, then predict the target using the processed data.

Such that, depending on the input patient data format: *tabular* or *sequence*, we have the following two subtasks: `indiv_outcome.tabular` and `indiv_outcome.sequence`.

2.5.1 Tabular Patient: Index

Here is the list of colab examples on each model for this task.

- Model 1: LogisticRegression
- Model 2: XGBoost
- Model 3: MLP
- Model 4: FTTransformer
- Model 5: TransTab

2.5.2 Sequential Patient: Index

Here is the list of colab examples on each model for this task.

- Model 1: RNN
- Model 2: RETAIN
- Model 3: RAIM
- Model 4: Dipole
- Model 5: StageNet

2.5.3 Tabular: Example

Here, we highlight the usage of `indiv_outcome.tabular.transtab` model for this task. Besides learning and predicting on a single tabular dataset, `transtab` shows promising performances on learning *across* different datasets. It sheds light on training a fundational model for clinical trials.

2.5.4 Sequence: Example

Here, we highlight the usage of `indiv_outcome.sequence.rnn` model for this task. Other more advanced models have the similar pipeline.

2.6 Clinical Trial Site Selection

2.7 Trial Outcome Prediction

Table of Contents

Making clinical trial outcome prediction is valuable at the planning stage. We can avoid running clinical trials that are probable to terminate early. Stakeholders can save unnecessary funding and time from those failing trials, reinvest in other promising trials or re-design those failing trials for a better outcome.

Formally, given the multi-aspect information of trials, e.g., drug molecules involved in this trial $\{m_1, m_2, \dots\}$, trial eligibility criteria $\{c_1, c_2 \dots\}$, target diseases $\{d_1, d_2, \dots\}$, predict the trial outcomes $y \in \{0, 1\}$ before the start of Phase I, where $y = 0$ indicates the trial failure and $y = 1$ indicates success. Or, add the trial patient records $X = \{x_1, x_2, \dots\}$ collected from previous stages, and predict the outcomes of Phase II or III $y \in \{0, 1\}$.

2.7.1 Trial Outcome Prediction: Index

Here is the list of colab examples on each model for this task.

- Model 1: LogisticRegression
- Model 2: XGBoost
- Model 3: MLP
- Model 4: HINT
- Model 5: SPOT

2.7.2 Trial Outcome Prediction: Example

Here, we highlight the usage of `trial_outcome.hint` model for this task.

TODO

2.8 Patient-Trial Matching

Table of Contents

The patient-trial matching is a process that identifies patients who are eligible to participate in a clinical trial. The matching process is based on the patient's electronic healthcare records (EHRs) and the trial's inclusion/exclusion criteria. To formulate it as a machine learning problem, we first need to define the input and output of the matching process.

A patient's EHR can be represented by a sequence of visits $V = \{v_1, v_2, ..., v_n\}$, where each visit v_i is a sequence of events $v_i = \{o_{i1}, o_{i2}, ..., o_{im}\}$. We can encode V to a compact embedding $E_p \in R^d$ as the patient's representation. The trial's inclusion/exclusion criteria can be represented by a sequence of inclusion/exclusion criteria $C = \{c_1, c_2, ..., c_m\}$. We can encode C to a compact embedding $E_c \in R^{m \times d}$ the trial's representation (each criterion is a vector). Such that, the neural matching process is just compute the similarity between E_p and E_c^m to get the criterion-level affinity.

2.8.1 Patient-Trial Matching: Index

Here is the list of colab examples on each model for this task.

- Model 1: DeepEnroll
- Model 2: COMPOSE

2.8.2 Patient-Trial Matching: Example

Here, we highlight the usage of `trial_patient_match.deepenroll` model for this task. Other models have the similar pipeline.

2.9 Trial Similarity Search

Table of Contents

We implement trial search task that aims to find similar trials to a given trial. This function is rather useful when practioners are designing new trials. They can refer to the retrieved historical trials, which provide valueable information for the trial's design, outcomes, and also look for potential collaborations. To be more specific, we concentrate on the *dense retrieval* where each trial document is encoded as a dense vector. Such that, trial similarity is measured through the cosine similarity between the trial vectors.

Formally, a trial document consists of multiple components, as $D = \{x^{title}, x^{intv}, x^{disc}, \dots\}$. The target of dense retrieval is to convert the input document to a dense vector $V \in R^d$, such that we can compute the trial's similarities and identify similar trials.

2.9.1 Trial Similarity Search: Index

- Model 1: Doc2Vec
- Model 2: WhitenBERT
- Model 3: Trial2Vec

2.9.2 Trial Similarity Search: Example

Here, we highlight the usage of `trial_search.trial2vec` model for this task. Other models have the similar pipeline.

2.10 Trial Patient Records Simulation

Table of Contents

Synthetic patient records generation is a way around privacy issues when sharing clinical trial records or healthcare data. Specifically, we want to train a generative model based on the real records, as $p(h^{syn}|h_1, h_2, \dots, h_n; \theta)$, where h^{syn} is the synthetic records, $h_1, h_2, \dots, h_n$ are the real records, and θ are the parameters of the model. When the patient data is a sequence, we can apply the generative model to the conditional generation. Given the previous visits $v_{1:t-1}$, we can generate the next visit record v_t as $v_t \sim p(v_t|v_{1:t-1}; \theta)$.

Depending ono the input patient data format: *tabular* or *sequence*, we have the following two subtasks: `trial_simulation.tabular` and `trial_simulation.sequence`.

2.10.1 Tabular Patient: Index

Here is the list of colab examples on each model for this task.

- Model 1: GaussianCopula
- Model 2: CopulaGAN
- Model 3: CTGAN
- Model 4: TVAE
- Model 5: MedGAN

2.10.2 Sequential Patient: Index

Here is the list of colab examples on each model for this task.

- Model 1: RNNGAN
- Model 2: EVA
- Model 3: SynTEG
- Model 4: PromptEHR
- Model 5: KNNSampler
- Model 6: TWIN

2.10.3 Tabular Patient: Example

Here, we highlight the usage of `trial_simulation.tabular.CTGAN` model for this task.

2.10.4 Sequential Patient: Example

Here, we highlight the usage of `trial_simulation.sequence.TWIN` model for this task.

2.11 Load Preprocessed Demo Data

In `PyTrial`, we provide a set of preprocessed demo data for quick start. The *data.demo_data* module provides a series of functions to load the demo data.

No need to download manually, the function will download the data automatically and save to the local disk, e.g.,

```
from pytrial.data.demo_data import load_synthetic_ehr_sequence

# load the demo data
data = load_synthetic_ehr_sequence()

# or specify the data path
data = load_synthetic_ehr_sequence(input_dir='./data')
```

Please refer to *data.demo_data* for a full list of available demo data.

2.12 Prepare Oncology Trial Patient Data

We provide an example of how to transform the raw patient-level records to sequence of patient data.

The jupyter notebook example is available at process_NCT00174655.ipynb.

The raw input data comes from Project Data Sphere (PDS) and is available at https://data.projectdatasphere.org/projectdatasphere/html/access. You need to create an account (it's free) and download the data from trial **NCT00174655**, put the raw data under the folder `./breast cancer/NCT00174655/`.

Note that the raw data are in SAS form `.sas7bdat`. We need to install `pip install sas7bdat` package to read the data.

2.13 Pretrained BERT Model

PyTrial provides an easy-to-use interface to load and use pretrained BERT model.

```
from pytrial.model_utils.bert import BERT

# Load pretrained BERT model
model = BERT()

# encode
emb = model.encode('The goal of life is comfort.')
```

The default pretrained BERT used is emilyalsentzer/Bio_ClinicalBERT. You can swith to other pretrained BERT models by specifying the model name as

```
# specify model name
model = BERT(bertname='bert-base-uncased')
```

The passed `bertname` should be one of the pretrained BERT models listed in HuggingFace Model Hub.

2.14 ICD9 & 10 Knowledge Graph

Table of Contents

2.14.1 ICD Knowledge Graph

PyTrial provides a way of creating ICD code graph and get the patient, children, and siblings of a given node.

The jupyter notebook example is available at demo_icd_graph.ipynb.

To be specific, this function is supported by `pytrial.model_utils.icd.ICD9Graph`, which returns an `nxgraph` property that is a `networkx.DiGraph` object.

```
from pytrial.model_utils.icd import ICD9Graph

# will download from the remote and save
graph = ICD9Graph()

from pytrial.model_utils.icd import ICD10Graph
graph = ICD10Graph(version='2021')
```

We can the children, parent, siblings of a node as

```
print(graph.children('Multiple gestation placenta status'))

print(graph.parent('Multiple gestation placenta status'))

print(graph.siblings('V9101'))
```

The `networkx.DiGraph` object can be used to do more complex graph analysis, for example, graph neural networks.

2.14.2 ICD Knowledge Query

We also provide a bunch of functions to query the ICD knowledge graph, available at *model_utils.icd*.

For example,

```
from pytrial.model_utils.icd import get_icd10_from_nih

print(get_icd10_from_nih('Multiple gestation placenta status'))
```

The function *get_icd10_from_nih* will return the ICD10 code of a given natural language description of a term. This function is supported by a public API provided by https://clinicaltables.nlm.nih.gov.

The other three available functions are: *get_icd9dx_from_nih*, *get_icd9sg_from_nih*, and *get_condition_synonym_from_nih*.

2.15 Drug Knowledge Graph

Table of Contents

2.15.1 Drug Transformer

PyTrial also offers utilities for processing and transforming drug related data. The first one is *pytrial.model_utils.drug.DrugTransformer* that works for mapping between different drug terminologies, e.g., from ATC to NDC, from ATC to SMILES, etc.

A colab example is available at demo_drug_utils.ipynb.

```
from pytrial.model_utils.drug import DrugTransformer

# Create a transformer
dt = DrugTransformer()

# drug name to SMILES
print(dt.name2smiles('Acetylsalicylic acid'))

# NDC to ATC4
print(dt.ndc2atc('00002140701'))

# ATC4 to NDC
print(dt.atc2ndc('C01BA'))

# drug name to NDC
print(dt.name2ndc('NEO*IV*Gentamicin'))

# NDC to drug name
print(dt.ndc2name('63323017302'))

# ATC4 to drug name
print(dt.atc2name('S01HA'))

# drug name to ATC4
print(dt.name2atc('atomoxetine'))

# NDC to SMILES
print(dt.ndc2smiles(['00002140701','00088222033']))

# ATC4 to SMILES
print(dt.atc2smiles(['S01HA','N06AX']))
```

2.15.2 Drug Graph

pytrial.model_utils.drug.DrugGraph is a class for creating a drug knowledge graph.

A colab example is available at demo_drug_graph.ipynb.

```
from pytrial.model_utils.drug import DrugGraph

dg = DrugGraph()

print(dg.graph)
'''
DiGraph with 251947 nodes and 303921 edges
'''
```

The yielded `dg.Graph` is a `networkx.DiGraph` object, which can be used to analyze the drug knowledge graph or create a graph neural network model.

CHAPTER

THREE

DATA

3.1 data.patient_data

class pytrial.data.patient_data.**TabularPatientBase**(*df*, *metadata=None*, *transform=True*)

Base dataset class for tabular patient records. Subclass it if additional properties and functions are required to add for specific tasks. We make use *rdt*: https://docs.sdv.dev/rdt for transform and reverse transform of the tabular data.

Parameters

- **df** (*pd.DataFrame*) – The input patient tabular format records.
- **metadata** – Contains the meta setups of the input data. It should contain the following keys:

(1) *sdtypes*: dict, the data types of each column in the input data. The keys are the column names and the values are the data types. The data types can be one of the following: 'numerical', 'categorical', 'datetime', 'boolean'.

(2) *transformers*: dict, the transformers to be used for each column. The keys are the column names and the values are the transformer names. The transformer names can be one in https://docs.sdv.dev/rdt/transformers-glossary/browse-transformers. In addition, we also support inputting a transformer string name, e.g., {'column1': 'OneHotEncoder'}.

transform: bool(default=True) Whether or not transform raw self.df by hypertransformer. If set False, `self.df` will keep as the same as the passed one.

Examples

```
>>> from pytrial.data.patient_data import TabularPatientBase
>>> df = pd.read_csv('tabular_patient.csv', index_col=0)
>>> # set `transform=True` will replace dataset.df with dataset.df_transformed
>>> dataset = TabularPatientBase(df, transform=True)
>>> # transform raw dataframe to numerical tables
>>> df_transformed = dataset.transform(df)
>>> # make back transform to the original df
>>> df_raw = dataset.reverse_transform(df_transformed)
```

reverse_transform(*df=None*)

Reverse the input dataframe back to the original format. Return the self.df in the original format if *df=None*.

Parameters **df** (*pd.DataFrame*) – The dataframe to be transformed back to the original format by self.ht.

transform(*df=None*)

Transform the input df or the self.df by hypertransformer. If transform=True in *__init__*, then you do not need to call this function to transform self.df because it was tranformed already.

Parameters **df** (*pd.DataFrame*) – The dataframe to be transformed by self.ht

class pytrial.data.patient_data.**SequencePatientBase**(*data*, *metadata=None*)

Load sequential patient inputs for longitudinal patient records generation.

Parameters

- **data** (*dict*) – A dict contains patient data in sequence and/or in tabular.

  data = {

  'x': np.ndarray or pd.DataFrame

  Static patient features in tabular form, typically those baseline information.

  'v': list or np.ndarray

  Patient visit sequence in dense format or in tensor format (depends on the model input requirement.)

  – If in dense format, it is like [[c1,c2,c3],[c4,c5],...], with shape [n_patient, NA, NA];

  – If in tensor format, it is like [[0,1,1],[1,1,0],...] (multi-hot encoded), with shape [n_patient, max_num_visit, max_num_event].

  'y': np.ndarray or pd.Series

  – Target label for each patient if making risk detection, with shape [n_patient, n_class];

  – Target label for each visit if making some visit-level prediction with shape [n_patient, NA, n_class].

  }

- **metadata** (*dict (optional)*) – A dict contains configuration of input patient data.

  metadata = {

  'voc': dict[Voc]

  Vocabulary contains the event index to the exact event name, has three keys in general: 'diag', 'med', 'prod', corresponding to diagnosis, medication, and procedure. `Voc` object should have two functions: *idx2word* and *word2idx*.

  'visit': dict[str]

  a dict contains the format of input data for processing input visit sequences.

  visit: {

  'mode': 'tensor' or 'dense',

  'order': list[str] (required when *mode='tensor'*)

  },

  'label': dict[str]

  a dict contains the format of input data for processing input labels.

  label: {

  'mode': 'tensor' or 'dense',

  }

'max_visit': int

the maximum number of visits considered when building tensor inputs, ignored when visit mode is dense.

}

class pytrial.data.patient_data.**SeqPatientCollator**(*config=None*)
support make collation of unequal sized list of batch for densely stored sequential visits data.

Parameters **config** (*dict*) – {

'visit_mode': in 'dense' or 'tensor',

'label_mode': in 'dense' or 'tensor',

}

3.2 data.trial_data

class pytrial.data.trial_data.**TrialDatasetBase**(*data*, *criteria_column='criteria'*)
The basic trial datasets loader.

Parameters

- **data** (*pd.DataFrame*) – Contain the trial document in tabular format.
- **criteria_column** (*str*) – The column name of eligibility criteria in the dataframe.

get_ec_sentence_embedding()
Process the eligibility criteria of each trial, get the criterion-level emebddings stored in dict.

Parameters **criteria_column** (*str*) – The column name of eligibility criteria in the dataframe.

class pytrial.data.trial_data.**TrialOutcomeDatasetBase**(*data*, *columns=None*)
Basic trial outcome datasets loader.

Parameters **data** (*pd.DataFrame*) – Contain the trial document in tabular format.

3.3 data.vocab_data

class pytrial.data.vocab_data.**Vocab**

add_sentence(*sentence*)
Add a list of words to the vocabulary. If one word is in the vocab, then ignore it. Otherwise, add it to the vocab.

Parameters **sentence** (*list[str]*) – A list of words.

property vocab
The vocabulary where key is the index and value is the word.

Returns **vocab**

Return type dict[int, str]

property words
All the words in the vocab.

Returns **words**

Return type list[str]

3.4 data.demo_data

pytrial.data.demo_data.**load_synthetic_ehr_sequence**(*input_dir=None*, *n_sample=None*)
Load synthetic EHR patient sequence data, which was generated by PromptEHR (https://arxiv.org/pdf/2211.01761.pdf).

Parameters

- **input_dir** (*str*) – The folder that stores the demo data. If None, we will download the demo data and save it to './demo_data/synthetic_ehr'. Make sure to remove this folder if it is empty.
- **n_sample** (*int*) – The number of samples we want to load. If None, all data will be loaded.

pytrial.data.demo_data.**load_trial_patient_sequence**(*input_dir=None*)
Load synthetic sequential trial patient records.

Parameters **input_dir** (*str*) – The folder that stores the demo data. If None, we will download the demo data and save it to './demo_data/demo_patient_sequence/trial'. Make sure to remove this folder if it is empty.

pytrial.data.demo_data.**load_trial_patient_tabular**(*input_dir=None*)
Load synthetic tabular trial patient records.

Parameters **input_dir** (*str*) – The folder that stores the demo data. If None, we will download the demo data and save it to './demo_data/demo_trial_patient_data'. Make sure to remove this folder if it is empty.

pytrial.data.demo_data.**load_trial_outcome_data**(*input_dir=None*, *phase='I'*, *split='train'*)
Load trial outcome prediction (TOP) benchmark data.

Parameters

- **input_dir** (*str*) – The folder that stores the demo data. If None, we will download the demo data and save it to './demo_data/demo_trial_data'. Make sure to remove this folder if it is empty.
- **phase** (*{'I','II','III'}*) – The phase of the trial data. Can be 'I', 'II', 'III'.
- **split** (*{'train', 'test', 'valid'}*) – The split of the trial data. Can be 'train', 'test', 'valid'.

pytrial.data.demo_data.**load_trial_document_data**(*input_dir=None*, *n_sample=None*, *source='preprocessed'*, *date='20221001'*)
Load trial document data obtained from ClinicalTrials.gov.

Parameters

- **input_dir** (*str*) – The folder that stores the demo data. If None, we will download the demo data and save it to ''./demo_data/demo_trial_document'. Make sure to remove this folder if it is empty.
- **n_sample** (*int*) – The number of samples we want to load. If None, all data will be loaded.
- **source** (*{'clinicaltrials.gov', 'preprocessed'}*) – The source of the data. If 'clinicaltrials.gov', we will download the raw data from that website and process it. If 'preprocessed', we will load the preprocessed data.

- **date** (*str*) – The date of the clinicaltrials.gov copy. Only valid when `source='clinicaltrials.gov'`.

`pytrial.data.demo_data.`**load_mimic_ehr_sequence**(*input_dir=None*, *n_sample=None*)

Load EHR patient sequence data, which needs to be accessed via https://physionet.org/content/mimiciii/1.4/.

Parameters

- **input_dir** (*str*) – The folder that stores the demo data. If None, we will look for the demo data in './demo_data/demo_patient_sequence/ehr'.
- **n_sample** (*int*) – The number of samples we want to load. If None, all data will be loaded.

CHAPTER

FOUR

TASKS.INDIV_OUTCOME

4.1 tasks.indiv_outcome.tabular

4.1.1 indiv_outcome.tabular.base

class pytrial.tasks.indiv_outcome.tabular.base.**TabularIndivBase**(*experiment_id='test'*)

Abstract class for all individual outcome predictions based on tabular patient data.

Parameters **experiment_id** (*str, optional (default = 'test')*) – The name of current experiment.

eval(*mode=False*)

Set the model in evaluation mode. Work similar to *model.eval()* in PyTorch.

Parameters **mode** (*bool, optional (default = False)*) – Whether to set the model in evaluation mode. `False` means the model is in evaluation mode. `True` means the model is in training mode.

abstract fit(*train_data*, *valid_data*)

Fit function needs to be implemented after subclass.

Parameters

- **train_data** (*Any*) – Training data.
- **valid_data** (*Any*) – Validation data.

abstract load_model(*checkpoint*)

Load the pretrained model from disk, needs to be implemented after subclass.

Parameters **checkpoint** (*str*) – The path to the checkpoint file.

abstract predict(*test_data*)

Prediction function needs to be implemented after subclass.

Parameters **test_data** (*Any*) – Testing data.

abstract save_model(*output_dir*)

Save the model to disk, needs to be implemented after subclass.

Parameters **output_dir** (*str*) – The path to the output directory.

train(*mode=True*)

Set the model in training mode. Work similar to *model.train()* in PyTorch.

Parameters **mode** (*bool, optional (default = True)*) – Whether to set the model in training mode. `False` means the model is in evaluation mode. `True` means the model is in training mode.

4.1.2 indiv_outcome.tabular.LogisticRegression

class pytrial.tasks.indiv_outcome.tabular.logistic_regression.**LogisticRegression**(*weight_decay=1*, *dual=False*, *epochs=100*, *experiment_id='test'*)

Bases: *pytrial.tasks.indiv_outcome.tabular.base.TabularIndivBase*

Implement Logistic Regression model for tabular individual outcome prediction in clinical trials. Now only support *binary classification*.

Parameters

- **weigth_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Like in support vector machines, smaller values specify weaker regularization.
- **dual** (*bool*) – Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer *dual=False* when *n_samples > n_features*.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **experiment_id** (*str, optional (default='test')*) – The name of current experiment. Decide the saved model checkpoint name.

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

Train logistic regression model to predict patient outcome with tabular input data.

Parameters

- **train_data** (*dict*) – {

  'x': TabularPatientBase or pd.DataFrame,

  'y': pd.Series or np.ndarray

  }

  – 'x' contain all patient features;

  – 'y' contain labels for each row.
- **valid_data** (*Ignored.*) – Not used, present heare for API consistency by convention.

load_model(*checkpoint=None*)

Save the learned logistic regression model to the disk.

Parameters **checkpoint** (*str or None*) –

- If a directory, the only checkpoint file *.model* will be loaded.
- If a filepath, will load from this file;
- If None, will load from *self.checkout_dir*.

predict(*test_data*)

Make prediction probability based on the learned model. Save to *self.result_dir*.

Parameters **test_data** (*dict*) – {

'x': TabularPatientBase or pd.DataFrame,

'y': pd.Series or np.ndarray

}

- 'x' contain all patient features;

- 'y' contain labels for each row. Ignored for prediction function.

Returns

ypred – The predicted probability for each patient.

- For binary classification, return shape (n,);
- For multiclass classification, return shape (n, n_class).

Return type np.ndarray

save_model(*output_dir=None*)

Save the learned logistic regression model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

4.1.3 indiv_outcome.tabular.XGBoost

class pytrial.tasks.indiv_outcome.tabular.xgboost.**XGBoost**(*mode*, *n_estimators=100*, *max_depth=8*, *n_jobs=0*, *reg_alpha=0*, *reg_lambda=0*, *num_class=None*, *experiment_id='test'*)

Implement XGBoost model for tabular individual outcome prediction in clinical trials.

Parameters

- **n_estimators** (*int*) – Number of boosting rounds.
- **max_depth** (*int*) – Maximum tree depth for base learners.
- **mode** (*str*) – The task's objectives, in *binary*, *multiclass*, *multilabel*, or *regression*. Do not support early stopping when *multilabel*.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
- **reg_alpha** (*float*) – L1 regularization term on weights (xgb's alpha).
- **reg_lambda** (*float*) – L2 regularization term on weights (xgb's lambda).
- **experiment_id** (*str, optional (default='test')*) – The name of current experiment. Decide the saved model checkpoint name.

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

Train logistic regression model to predict patient outcome with tabular input data.

Parameters

- **train_data** (*dict*) – { 'x': TabularPatientBase or pd.DataFrame, 'y': pd.Series or np.ndarray }
  - 'x' contain all patient features;
  - 'y' contain labels for each row.
- **valid_data** (*same as train_data.*) – Validation set for early stopping.

load_model(*checkpoint=None*)

Load the learned XGBoost model from the disk.

Parameters **checkpoint** (*str or None*) –

- If a directory, the only checkpoint file *.model* will be loaded.

- If a filepath, will load from this file;
- If None, will load from *self.checkout_dir*.

predict(*test_data*)

Make prediction probability based on the learned model. Save to *self.result_dir*.

Parameters **test_data** (*dict*) – { 'x': TabularPatientBase or pd.DataFrame, 'y': pd.Series or np.ndarray }

- 'x' contain all patient features;
- 'y' contain labels for each row. Ignored for prediction function.

Returns **ypred** – For binary classification, return shape (n,); For multiclass classification, return shape (n, n_class).

Return type np.ndarray

save_model(*output_dir=None*)

Save the learned XGBoost model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

4.1.4 indiv_outcome.tabular.MLP

class pytrial.tasks.indiv_outcome.tabular.mlp.**MLP**(*input_dim*, *output_dim*, *mode*, *hidden_dim=128*, *num_layer=2*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)

Implement multi-layer perceptron model for tabular individual outcome prediction in clinical trials.

Parameters

- **input_dim** (*int*) – Dimension of the input features.
- **output_dim** (*int*) – Dimension of the outputs. When doing classification, it equals to number of classes.
- **mode** (*str*) – The task's objectives, in *binary*, *multiclass*, *multilabel*, or *regression*
- **hidden_dim** (*int*) – Hidden dimensions of neural networks.
- **num_layer** (*int*) – Number of hidden layers.
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **weigth_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – Target device to train the model, as *cuda:0* or *cpu*.
- **experiment_id** (*str, optional (default='test')*) – The name of current experiment. Decide the saved model checkpoint name.

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

Train logistic regression model to predict patient outcome with tabular input data.

Parameters

- **train_data** (*dict*) – { 'x': TabularPatientBase or pd.DataFrame, 'y': pd.Series or np.ndarray }
  - 'x' contain all patient features;
  - 'y' contain labels for each row.
- **valid_data** (same as *train_data*.) – Validation data during the training for early stopping.

load_model(*checkpoint*)

Load model and the pre-encoded trial embeddings from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model. - If a directory, the only checkpoint file **.pth.tar* will be loaded. - If a filepath, will load from this file.

predict(*test_data*)

Make prediction probability based on the learned model.

Parameters **test_data** (*Dict or TabularPatientBase or pd.DataFrame or torch.Tensor*) – {'x': TabularPatientBase or pd.DataFrame}

'x' contain all patient features.

Returns

ypred – Prediction probability for each patient.

- For binary classification, return shape (n,);
- For multiclass classification, return shape (n, n_class).

Return type np.ndarray or torch.Tensor

save_model(*output_dir=None*)

Save the learned logistic regression model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

4.1.5 indiv_outcome.tabular.FTTransformer

class pytrial.tasks.indiv_outcome.tabular.ft_transformer.**FTTransformer**(*num_feat*, *cat_feat*, *cat_cardinalities*, *output_dim*, *mode*, *hidden_dim=128*, *num_layer=2*, *attention_dropout=0*, *ffn_dim=256*, *ffn_dropout=0*, *residual_dropout=0*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)

Bases: *pytrial.tasks.indiv_outcome.tabular.base.TabularIndivBase*

Implement ft-transformer model for tabular individual outcome prediction in clinical trials[1].

Parameters

- **num_feat** (*list[str]*) – the list of numerical feature names.
- **cat_feat** (*list[str]*) – the list of categorical feature names.
- **cat_cardinalities** (*list[int]*) – A list of categorical features' cardinalities.
- **output_dim** (*int*) – Dimension of the outputs. When doing classification, it equals to number of classes.
- **mode** (*str*) – The task's objectives, in *binary*, *multiclass*, *multilabel*, or *regression*
- **hidden_dim** (*int*) – Hidden dimensions of neural networks. Must be a multiple of `n_heads=8`.
- **num_layer** (*int*) – Number of hidden layers.
- **attention_dropout** (*float*) – the dropout for attention blocks. Usually, positive values work better (even when the number of features is low).
- **ffn_dim** (*int*) – the *input* size for the *second* linear layer in *Transformer.FFN*. Note that it can be different from the output size of the first linear layer, since activations such as ReGLU or GEGLU change the size of input. For example, if `ffn_d_hidden=10` and the activation is ReGLU (which is always true for the baseline and default configurations), then the output size of the first linear layer will be set to `20`.
- **ffn_dropout** (*float*) – the dropout rate after the first linear layer in *Transformer.FFN*.
- **residual_dropout** (*float*) – the dropout rate for the output of each residual branch of all Transformer blocks.
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **weight_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.

[1] Gorishniy, Y., et al. (2021). Revisiting deep learning models for tabular data. NeurIPS'21.

- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – Target device to train the model, as *cuda:0* or *cpu*.
- **experiment_id** (*str, optional (default='test')*) – The name of current experiment. Decide the saved model checkpoint name.

Notes

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

Train FT-Transformer model to predict patient outcome with tabular input data.

Parameters

- **train_data** (*dict*) – { 'x': TabularPatientBase or pd.DataFrame, 'y': pd.Series or np.ndarray }
  - 'x' contain all patient features;
  - 'y' contain label for each row.
- **valid_data** (*dict*) – Same as *train_data*. Validation data during the training for early stopping.

load_model(*checkpoint*)

Load the model from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model.

- If a directory, the only checkpoint file **.pth.tar* will be loaded.
- If a filepath, will load from this file.

predict(*test_data*)

Make prediction probability based on the learned model.

Parameters **test_data** (*Dict or TabularPatientBase or pd.DataFrame or torch.Tensor*) – {'x': TabularPatientBase or pd.DataFrame or torch.Tensor}

'x' contain all patient features.

Returns

ypred – Prediction probability for each patient.

- For binary classification, return shape (n,);
- For multiclass classification, return shape (n, n_class).

Return type np.ndarray or torch.Tensor

save_model(*output_dir=None*)

Save the learned ft-transformer model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

4.1.6 indiv_outcome.tabular.TransTab

class pytrial.tasks.indiv_outcome.tabular.transtab.**TransTab**(*mode=None*, *categorical_columns=None*, *numerical_columns=None*, *binary_columns=None*, *contrastive_pretrain=False*, *num_class=2*, *hidden_dim=128*, *num_layer=2*, *num_attention_head=8*, *hidden_dropout_prob=0*, *ffn_dim=256*, *activation='relu'*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)

Bases: *pytrial.tasks.indiv_outcome.tabular.base.TabularIndivBase*

Implement transtab model for tabular individual outcome prediction in clinical trials[1].

Parameters

- **mode** (*str*) – The task's objectives, in *binary*, *multiclass*. # TODO: *multilabel*, or *regression* Can be ignored if *contrastive_pretrain* is set True.
- **categorical_columns** (*list*) – a list of categorical feature names.
- **numerical_columns** (*list*) – a list of numerical feature names.
- **binary_columns** (*list*) – a list of binary feature names, accept binary indicators like (yes,no); (true,false); (0,1).
- **contrastive_pretrain** (*bool(default=False)*) – whether or not take a contrastive pretraining. If set true, *num_class* will be ignored.
- **num_class** (*int*) – number of output classes to be predicted.
- **hidden_dim** (*int*) – the dimension of hidden embeddings.
- **num_layer** (*int*) – the number of transformer layers used in the encoder.
- **num_attention_head** (*int*) – the numebr of heads of multihead self-attention layer in the transformers.
- **hidden_dropout_prob** (*float*) – the dropout ratio in the transformer encoder.
- **ffn_dim** (*int*) – the dimension of feed-forward layer in the transformer layer.
- **activation** (*str*) – the name of used activation functions, support "relu", "gelu", "selu", "leakyrelu".
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **weight_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.

[1] Wang, Z., & Sun, J. (2022). TransTab: Learning Transferable Tabular Transformers Across Tables. NeurIPS'22.

- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – Target device to train the model, as *cuda:0* or *cpu*.
- **experiment_id** (*str, optional (default='test')*) – The name of current experiment. Decide the saved model checkpoint name.

Notes

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

Train TransTab model to predict patient outcome with tabular input data.

Parameters

- **train_data** (*list[dict]*) – a list of patient data, each patient is a dict of {

  'x': TabularPatientBase or pd.DataFrame,

  'y': pd.Series or np.ndarray

  }.

  TransTab can learn from multiple different tabular datasets.
- **valid_data** (*dict*) – Validation data during the training for early stopping. valid_data =

  {

  'x': TabularPatientBase or pd.DataFrame,

  'y': pd.Series or np.ndarray

  }

load_model(*checkpoint*)

Load the learned transtab model from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model.

- If a directory, the only checkpoint file **.pth.tar* will be loaded.
- If a filepath, will load from this file.

predict(*test_data*)

Make prediction probability based on the learned model.

Parameters **test_data** (TabularPatientBase *or pd.DataFrame*) – Contain all patient features.

Returns

ypred –

- For binary classification, return shape (n,);
- For multiclass classification, return shape (n, n_class).

Return type np.ndarray or torch.Tensor

save_model(*output_dir=None*)

Save the learned transtab model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

update(*config*)
Update the configuration of feature extractor's column map for *cat*, *num*, and *bin* cols. Or update the number of classes for the output classifier layer.

Parameters **config** (*dict*) – a dict of configurations: keys *cat:list*, *num:list*, *bin:list* are to specify the new column names; key *num_class:int* is to specify the number of classes for finetuning on a new dataset.

4.2 tasks.indiv_outcome.sequence

4.2.1 indiv_outcome.sequence.SequenceIndivBase

class pytrial.tasks.indiv_outcome.sequence.base.**SequenceIndivBase**(*experiment_id='test'*, *mode=None*, *output_dim=None*)

Abstract class for all individual outcome predictions based on sequential patient data.

Parameters **experiment_id** (*str, optional (default = 'test')*) – The name of current experiment.

eval(*mode=False*)
Swith the model to the *validation* mode. Work samely as *model.eval()* in pytorch.

Parameters **mode** (*bool, optional (default = False)*) – If False, switch to the *validation* mode.

abstract fit(*train_data*, *valid_data*)
Fit function needs to be implemented after subclass.

Parameters

- **train_data** (*Any*) – Training data.
- **valid_data** (*Any*) – Validation data.

abstract load_model(*checkpoint*)
Load the pretrained model from disk, needs to be implemented after subclass.

Parameters **checkpoint** (*str*) – The path to the checkpoint file.

abstract predict(*test_data*)
Prediction function needs to be implemented after subclass.

Parameters **test_data** (*Any*) – Testing data.

abstract save_model(*output_dir*)
Save the model to disk, needs to be implemented after subclass.

Parameters **output_dir** (*str*) – The path to the output directory.

train(*mode=True*)
Swith the model to the *training* mode. Work samely as *model.train()* in pytorch.

Parameters **mode** (*bool, optional (default = True)*) – If True, switch to the *training* mode.

4.2.2 indiv_outcome.sequence.RNN

class pytrial.tasks.indiv_outcome.sequence.rnn.**RNN**(*vocab_size*, *orders*, *mode*, *output_dim=None*, *max_visit=20*, *emb_size=64*, *n_rnn_layer=2*, *rnn_type='lstm'*, *bidirectional=False*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *evaluation_steps=100*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)

Implement an RNN based model for longitudinal patient records predictive modeling.

Parameters

- **vocab_size** (*list[int]*) – A list of vocabulary size for different types of events, e.g., for diagnosis, procedure, medication.
- **orders** (*list[str]*) – A list of orders when treating inputs events. Should have the same shape of *vocab_size*.
- **output_dim** (*int*) – Output dimension of the model.
  - If binary classification, output_dim=1;
  - If multiclass/multilabel classification, output_dim=n_class
  - If regression, output_dim=1.
- **mode** (*str*) – Prediction traget in ['binary','multiclass','multilabel','regression'].
- **max_visit** (*int*) – The maximum number of visits for input event codes.
- **emb_size** (*int*) – Embedding size for encoding input event codes.
- **n_rnn_layer** (*int*) – Number of RNN layers for encoding historical events.
- **rnn_type** (*str*) – Pick RNN types in ['rnn','lstm','gru']
- **bidirectional** (*bool*) – If True, it encodes historical events in bi-directional manner.
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **weigth_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **evaluation_steps** (*int*) – Number of steps to evaluate the model on validation set or report the training loss.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – The model device.
- **experiment_id** (*str*) – The prefix when saving the checkpoints during the training.

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

Train model with sequential patient records.

Parameters

- **train_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events; 'y' corresponds to labels.

- **valid_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records used to make early stopping of the model.

load_model(*checkpoint*)
Load pretrained model from the disk.

Parameters **checkpoint** (*str*) – The input directory that stores the trained pytorch model and configuration.

predict(*test_data*)
Predict patient outcomes using longitudinal trial patient sequences.

Parameters **test_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events.

save_model(*output_dir*)
Save the pretrained model to the disk.

Parameters **output_dir** (*str*) – The output directory that stores the trained pytorch model and configuration.

4.2.3 indiv_outcome.sequence.RETAIN

class pytrial.tasks.indiv_outcome.sequence.retain.**RETAIN**(*vocab_size*, *orders*, *mode*, *output_dim=None*, *max_visit=None*, *emb_size=64*, *n_rnn_layer=2*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)
Implement RETAIN for longitudinal patient records predictive modeling[1].

Parameters

- **vocab_size** (*list[int]*) – A list of vocabulary size for different types of events, e.g., for diagnosis, procedure, medication.
- **orders** (*list[str]*) – A list of orders when treating inputs events. Should have the same shape of *vocab_size*.
- **output_dim** (*int*) – The dimension of the output.
  - If binary classification, output_dim=1;
  - If multiclass/multilabel classification, output_dim=n_class
  - If regression, output_dim=1.
- **mode** (*str*) – Prediction traget in ['binary','multiclass','multilabel','regression'].
- **max_visit** (*int*) – The maximum number of visits for input event codes.
- **emb_size** (*int*) – Embedding size for encoding input event codes.
- **n_rnn_layer** (*int*) – Number of RETAIN layers for encoding historical events.
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.

[1] Choi, E., Bahadori, M. T., Sun, J., Kulas, J., Schuetz, A., & Stewart, W. (2016). Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. Advances in neural information processing systems, 29.

- **weight_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – The model device.

Notes

fit(*train_data*, *valid_data=None*)
Train model with sequential patient records.

Parameters

- **train_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events; 'y' corresponds to labels.
- **valid_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records used to make early stopping of the model.

load_model(*checkpoint*)
Load pretrained model from the disk.

Parameters **checkpoint** (*str*) – The input directory that stores the trained pytorch model and configuration.

predict(*test_data*)
Predict patient outcomes using longitudinal trial patient sequences.

Parameters **test_data** (*SequencePatient*) – A *SequencePatient* contains patient records where 'v' corresponds to visit sequence of different events.

save_model(*output_dir*)
Save the pretrained model to the disk.

Parameters **output_dir** (*str*) – The output directory that stores the trained pytorch model and configuration.

4.2.4 indiv_outcome.sequence.RAIM

class pytrial.tasks.indiv_outcome.sequence.raim.**RAIM**(*vocab_size*, *orders*, *mode=None*, *output_dim=None*, *max_visit=None*, *window_size=3*, *hidden_size=8*, *hidden_output_size=8*, *dropout=0.5*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)
Implment RAIM for longitudinal patient records predictive modeling[1].

Parameters

- **vocab_size** (*list[int]*) – A list of vocabulary size for different types of events, e.g., for diagnosis, procedure, medication.

[1] Xu, Y., Biswal, S., Deshpande, S. R., Maher, K. O., & Sun, J. (2018, July). Raim: Recurrent attentive and intensive model of multimodal patient monitoring data. In Proceedings of the 24th ACM SIGKDD international conference on Knowledge Discovery & Data Mining (pp. 2565-2573).

- **orders** (*list[str]*) – A list of orders when treating inputs events. Should have the same shape of *vocab_size*.
- **mode** (*str*) – Prediction traget in ['binary','multiclass','multilabel','regression'].
- **output_dim** (*int*) – Output dimension of the model.
  - If binary classification, output_dim=1;
  - If multiclass/multilabel classification, output_dim=n_class;
  - If regression, output_dim=1.
- **max_visit** (*int*) – The maximum number of visits for input event codes.
- **window_size** (*int*) – The window size in the recurrent part of RAIM.
- **hidden_size** (*int*) – Embedding size for encoding input event codes.
- **hidden_output_size** (*int*) – The output size of the intermediate layers. Not the output_dim.
- **dropout** (*float*) – Dropout rate in the intermediate layers.
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **weight_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – The model device.
- **experiment_id** (*str*) – The prefix when saving the checkpoints during the training.

Notes

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

Train model with sequential patient records.

Parameters

- **train_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events; 'y' corresponds to labels.
- **valid_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records used to make early stopping of the model.

load_model(*checkpoint*)

Load pretrained model from the disk.

Parameters **checkpoint** (*str*) – The input directory that stores the trained pytorch model and configuration.

predict(*test_data*)

Predict patient outcomes using longitudinal trial patient sequences.

Parameters **test_data** (*SequencePatient*) – A *SequencePatient* contains patient records where 'v' corresponds to visit sequence of different events.

save_model(*output_dir*)
Save the pretrained model to the disk.

Parameters **output_dir** (*str*) – The output directory that stores the trained pytorch model and configuration.

4.2.5 indiv_outcome.sequence.Dipole

class pytrial.tasks.indiv_outcome.sequence.dipole.**Dipole**(*vocab_size*, *orders*, *mode*, *output_dim=None*, *max_visit=None*, *attention_type='location_based'*, *attention_dim=8*, *emb_size=16*, *hidden_size=8*, *hidden_output_size=8*, *dropout=0.5*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)

Implement Dipole for longitudinal patient records predictive modeling[1].

Parameters

- **vocab_size** (*list[int]*) – A list of vocabulary size for different types of events, e.g., for diagnosis, procedure, medication.
- **orders** (*list[str]*) – A list of orders when treating inputs events. Should have the same shape of *vocab_size*.
- **mode** (*str*) – Prediction traget in ['binary','multiclass','multilabel','regression'].
- **output_dim** (*int*) – Output dimension of the model.
  - If binary classification, output_dim=1;
  - If multiclass/multilabel classification, output_dim=n_class
  - If regression, output_dim=1.
- **max_visit** (*int*) – The maximum number of visits for input event codes.
- **attention_type** (*{'general', 'concatenation_based', 'location_based'}*) – Apply attention mechnism to derive a context vector that captures relevant information to help predict target.
  - 'location_based': Location-based Attention. Alocation-based attention function is to calculate the weights solely from hidden state
  - 'general': General Attention. An easy way to capture the relationship between two hidden states
  - 'concatenation_based': Concatenation-based Attention. Via concatenating two hidden states, then use multi-layer perceptron(MLP) to calculatethe contextvector
- **attention_dim** (*int*) – It is the latent dimensionality used for attention weight computing just for for concatenation_based attention mechnism
- **emb_size** (*int*) – Embedding size for encoding input event codes.
- **hidden_size** (*int, optional (default = 8)*) – The number of features of the hidden state h

[1] Ma, F., Chitta, R., Zhou, J., You, Q., Sun, T., & Gao, J. (2017, August). Dipole: Diagnosis prediction in healthcare via attention-based bidirectional recurrent neural networks. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1903-1911).

- **hidden_output_size** (*int, optional (default = 8)*) – The number of mix features
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **weight_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – The model device.

Notes

fit(*train_data*, *valid_data*)
Train model with sequential patient records.

Parameters

- **train_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events; 'y' corresponds to labels.
- **valid_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records used to make early stopping of the model.

load_model(*checkpoint*)
Load pretrained model from the disk.

Parameters **checkpoint** (*str*) – The input directory that stores the trained pytorch model and configuration.

predict(*test_data*)
Predict patient outcomes using longitudinal trial patient sequences.

Parameters **test_data** (*SequencePatient*) – A *SequencePatient* contains patient records where 'v' corresponds to visit sequence of different events.

save_model(*output_dir*)
Save the pretrained model to the disk.

Parameters **output_dir** (*str*) – The output directory that stores the trained pytorch model and configuration.

4.2.6 indiv_outcome.sequence.StageNet

class pytrial.tasks.indiv_outcome.sequence.stagenet.**StageNet**(*vocab_size*, *orders*, *mode*, *output_dim=None*, *max_visit=None*, *hidden_size=384*, *conv_size=10*, *levels=3*, *dropconnect=0.3*, *dropout=0.3*, *dropres=0.3*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)

Implement StageNet for longitudinal patient records predictive modeling[1].

Parameters

- **vocab_size** (*list[int]*) – A list of vocabulary size for different types of events, e.g., for diagnosis, procedure, medication.
- **orders** (*list[str]*) – A list of orders when treating inputs events. Should have the same shape of *vocab_size*.
- **mode** (*str*) – Prediction traget in ['binary','multiclass','multilabel','regression'].
- **output_dim** (*int*) – If binary classification, output_dim=1; If multiclass/multilabel classification, output_dim=n_class If regression, output_dim=1.
- **max_visit** (*int*) – The maximum number of visits for input event codes.
- **hidden_size** (*int, optional (default = 8)*) – The number of features of the hidden state h.
- **conv_size** (*int, optional (default = 10)*) – The number of convolution kernels.
- **levels** (*int, optional (default = 3)*) – The number of levels for the master gate.
- **dropconnect** (*float, optional (default = 0.3)*) – The dropout rate for the input of the convolutional layer.
- **dropout** (*float, optional (default = 0.3)*) – The dropout rate for the output of the RNN layer.
- **dropres** (*float, optional (default = 0.3)*) – The dropout rate for the residual connection.
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **weight_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – The model device.

[1] Gao, J., Xiao, C., Wang, Y., Tang, W., Glass, L. M., & Sun, J. (2020, April). Stagenet: Stage-aware neural networks for health risk prediction. In Proceedings of The Web Conference 2020 (pp. 530-540).

Notes

fit(*train_data*, *valid_data*)
Train model with sequential patient records.

Parameters

- **train_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events; 'y' corresponds to labels.
- **valid_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records used to make early stopping of the model.

load_model(*checkpoint*)
Load pretrained model from the disk.

Parameters **checkpoint** (*str*) – The input directory that stores the trained pytorch model and configuration.

predict(*test_data*)
Predict patient outcomes using longitudinal trial patient sequences.

Parameters **test_data** (*SequencePatient*) – A *SequencePatient* contains patient records where 'v' corresponds to visit sequence of different events.

save_model(*output_dir*)
Save the pretrained model to the disk.

Parameters **output_dir** (*str*) – The output directory that stores the trained pytorch model and configuration.

CHAPTER

FIVE

TASKS.SITE_SELECTION

5.1 site_selection.SiteSelectionBase

class pytrial.tasks.site_selection.base.**SiteSelectionBase**(*experiment_id='test'*)

Abstract class for all sequential patient data simulations.

Parameters **experiment_id** (*str, optional (default = 'test')*) – The name of current experiment.

eval(*mode=False*)

Swith the model to the *validation* mode. Work samely as *model.eval()* in pytorch.

Parameters **mode** (*bool, optional (default = False)*) – If False, switch to the *validation* mode.

abstract fit(*train_data*)

Fit function needs to be implemented after subclass.

Parameters **train_data** (*Any*) – Training data.

abstract load_model(*checkpoint*)

Load the pretrained model from disk, needs to be implemented after subclass.

Parameters **checkpoint** (*str*) – The path to the checkpoint file.

abstract predict(*test_data*)

Prediction function needs to be implemented after subclass.

Parameters **test_data** (*Any*) – Testing data.

abstract save_model(*output_dir*)

Save the model to disk, needs to be implemented after subclass.

Parameters **output_dir** (*str*) – The path to the output directory.

train(*mode=True*)

Swith the model to the *training* mode. Work samely as *model.train()* in pytorch.

Parameters **mode** (*bool, optional (default = True)*) – If True, switch to the *training* mode.

5.2 site_selection.PolicyGradientEntropy

class pytrial.tasks.site_selection.pgentropy.**PolicyGradientEntropy**(*trial_dim=211*, *site_dim=124*, *embedding_dim=64*, *enrollment_only=True*, *K=10*, *lam=1*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)

Implement Policy Gradient Entropy model for selecting clinical trial sites based on possibly missing multi-model site features.[1]

Parameters

- **trial_dim** (*list[int]*) – Size of the trial representation
- **site_dim** (*int*) – Size of the site representation
- **embedding_dim** (*int*) – Size of all of the modality and other intermediate embeddings

Notes

fit(*train_data*)

Train model with historical trial-site enrollments.

Parameters **train_data** (*TrialSiteSimple*) – A *TrialSiteSimple* contains trials, sites, and enrollments.

load_model(*checkpoint*)

Load model and the pre-encoded trial embeddings from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model. If a directory, the only checkpoint file **.pth.tar* will be loaded. If a filepath, will load from this file.

predict(*test_data*)

Make prediction for site selection.

save_model(*output_dir*)

Save the learned patient-match model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

[1] Srinivasa, R. S., Qian, C., Theodorou, B., Spaeder, J., Xiao, C., Glass, L., & Sun, J. (2022). Clinical trial site matching with improved diversity using fair policy learning. arXiv preprint arXiv:2204.06501.

CHAPTER

SIX

TASKS.TRIAL_OUTCOME

6.1 trial_outcome.LogisticRegression

class pytrial.tasks.trial_outcome.logistic_regression.**LogisticRegression**(*C=1.0*, *dual=False*, *solver='lbfgs'*, *max_iter=100*)

Implement Logistic Regression model for clinical trial outcome prediction.

Parameters

- **C** (*float*) – Regularization strength for l2 norm; must be a positive float. Like in support vector machines, smaller values specify weaker regularization.
- **dual** (*bool*) – Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer *dual=False* when *n_samples* > *n_features*.
- **solver** (*{'newton-cg','lbfgs','liblinear','sag','saga'}*) – Algorithm to use in the optimization problem. default='lbfgs'.
- **max_iter** (*int (default=100)*) – Maximum number of iterations taken for the solvers to converge.

fit(*train_data*, *valid_data*)

Train logistic regression model to predict clinical trial outcomes.

Parameters

- **train_data** (TrialOutcomeDatasetBase) – Training data, should be a *TrialOutcomeDatasetBase* object.
- **valid_data** (TrialOutcomeDatasetBase) – Validation data, should be a *TrialOutcomeDatasetBase* object. Ignored for logistic regression model. Keep this parameter for compatibility with other models.

load_model(*checkpoint=None*)

Load the learned model from the disk.

Parameters **checkpoint** (*str*) – The checkpoint folder to load the learned model. The checkpoint under this folder should be *model.ckpt*.

predict(*test_data*)

Make clinical trial outcome predictions.

Parameters **test_data** (TrialOutcomeDatasetBase) – Testing data, should be a *TrialOutcomeDatasetBase* object.

save_model(*output_dir=None*)

Save the learned logistic regression model to the disk.

Parameters **output_dir** (*str or None*) – The output folder to save the learned model. If set None, will save model to *save_model/model.ckpt*.

6.2 trial_outcome.MLP

class pytrial.tasks.trial_outcome.mlp.**MLP**(*epoch=5*, *lr=0.001*, *batch_size=32*, *weight_decay=0.0*)

Implement MLP model for clinical trial outcome prediction.

Parameters

- **epoch** (*int*) – number of training epochs.
- **lr** (*float*) – learning rate.
- **weight_decay** (*float*) – Regularization strength for l2 norm; must be a positive float.

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

Train model to predict clinical trial outcomes.

Parameters

- **train_data** (TrialOutcomeDatasetBase) – Training data, should be a *TrialOutcomeDatasetBase* object.
- **valid_data** (TrialOutcomeDatasetBase) – Validation data, should be a *TrialOutcomeDatasetBase* object.

load_model(*checkpoint=None*)

Load the learned MLP model from the disk.

Parameters **checkpoint** (*str*) – The checkpoint folder to load the learned model. The checkpoint under this folder should be *model.ckpt*.

predict(*test_data*)

Make clinical trial outcome predictions.

Parameters **test_data** (TrialOutcomeDatasetBase) – Testing data, should be a *TrialOutcomeDatasetBase* object.

save_model(*output_dir=None*)

Save the learned model to the disk.

Parameters **output_dir** (*str or None*) – The output folder to save the learned model. If set None, will save model to *save_model/model.ckpt*.

6.3 trial_outcome.XGBoost

class pytrial.tasks.trial_outcome.xgboost.**XGBoost**(*n_estimators=100*, *max_depth=3*, *reg_lambda=0*, *eval_metric='auc'*)

Implement XGBoost model for clinical trial outcome prediction.

Parameters

- **n_estimators** (*int*) – number of trees in the forest
- **max_depth** (*int*) – maximum depth of the tree
- **reg_lambda** (*float*) – L2 regularization term on weights
- **eval_metric** (*{'auc', 'logloss'}*) – evaluation metric for validation data, default is AUC.

fit(*train_data*, *valid_data=None*)
Train model to predict clinical trial outcomes.

Parameters

- **train_data** (TrialOutcomeDatasetBase) – Training data, should be a *TrialOutcomeDatasetBase* object.
- **valid_data** (TrialOutcomeDatasetBase) – Validation data, should be a *TrialOutcomeDatasetBase* object.

load_model(*checkpoint=None*)
Load the learned model from the disk.

Parameters **checkpoint** (*str*) – The checkpoint folder to load the learned model. The checkpoint under this folder should be *model.ckpt*.

predict(*test_data*)
Make clinical trial outcome predictions.

Parameters **test_data** (TrialOutcomeDatasetBase) – Testing data, should be a *TrialOutcomeDatasetBase* object.

save_model(*output_dir=None*)
Save the learned model to the disk.

Parameters **output_dir** (*str or None*) – The output folder to save the learned model. If set None, will save model to *save_model/model.ckpt*.

6.4 trial_outcome.HINT

class pytrial.tasks.trial_outcome.hint.**HINT**(*disease_embedding_dim=50*, *protocol_output_dim=50*, *molecule_embedding_dim=50*, *global_embed_size=50*, *highway_num_layer=3*, *gnn_hidden_size=50*, *epoch=20*, *lr=0.0003*, *batch_size=32*, *weight_decay=0*, *prefix_name='phase_I'*, *device='cuda:0'*)
Implement Hierarchical Interaction Network (HINT) model for clinical trial outcome prediction[1].

Parameters

- **disease_embedding_dim** (*int*) – dimension of disease code embedding, e.g., 50
- **protocol_output_dim** (*int*) – dimension of protocol (eligibility criteria) embedding, e.g., 50
- **molecule_embedding_dim** (*int*) – dimension of molecule embedding, e.g., 50
- **global_embed_size** (*int*) – dimension of trial component embedding, e.g., 50
- **highway_num_layer** (*int*) – number of highway layers, e.g., 3
- **gnn_hidden_size** (*int*) – dimension of GNN hidden size, e.g., 50
- **epoch** (*int*) – epoch number during training, e.g., 5
- **lr** (*float*) – learning rate of optimizer (we use Adam) during training, e.g., 3e-4,
- **batch_size** (*int*) – batch size during training, e.g., 32
- **weight_decay** (*float*) – weight decay coefficient, e.g., 0.

[1] Fu et al. HINT: Hierarchical Interaction Network for Clinical Trial Outcome Prediction. Cell Patterns, 2022.

- **prefix_name** (*str*) – name of trial phase as prefix name of the model, e.g., *phase_I*, *phase_II*
- **device** (*str or torch.device*) – Target device to train the model, as *cuda:0* or *cpu*.

Notes

fit(*train_data*, *valid_data=None*)
Train HINT model to predict clinical trial outcome (approval rate)

Parameters

- **train_data** (TrialOutcomeDatasetBase) – Training data, should be a *TrialOutcomeDatasetBase* object.
- **valid_data** (TrialOutcomeDatasetBase) – Validation data, should be a *TrialOutcomeDatasetBase* object.

load_model(*checkpoint=None*)
Load the learned HINT model from the disk.

Parameters **checkpoint** (*str*) – The checkpoint folder to load the learned model. The checkpoint under this folder should be *model.ckpt*.

predict(*test_data*)
Make trial outcome prediction for test data.

Parameters **test_data** (TrialOutcomeDatasetBase) – Testing data, should be a *TrialOutcomeDatasetBase* object.

save_model(*output_dir=None*)
Save the learned HINT model to the disk.

Parameters **output_dir** (*str or None*) – The output folder to save the learned model. If set None, will save model to *checkpoints/model.ckpt*.

6.5 trial_outcome.SPOT

class pytrial.tasks.trial_outcome.spot.**SPOT**(*num_topics=50*, *n_trial_projector=2*, *n_timestemp_projector=2*, *n_rnn_layer=1*, *criteria_column='criteria'*, *batch_size=1*, *n_trial_per_batch=None*, *learning_rate=0.0001*, *weight_decay=0.0001*, *epochs=10*, *evaluation_steps=50*, *warmup_ratio=0*, *device='cuda:0'*, *seed=42*, *output_dir='./checkpoints/spot'*)
Implement sequential predictive modeling of clinical trial outcome with meta-learning (SPOT)[1].

Parameters

- **num_topics** (*int*) – Number of topics to discover.
- **n_trial_projector** (*int*) – Number of layers in the trial projector.
- **n_timestemp_projector** (*int*) – Number of layers in the timestamp projector.
- **n_rnn_layer** (*int*) – Number of layers in the RNN for encoding ontology & smiles strings.

[1] Wang, Z., Xiao, C., & Sun, J. (2023). SPOT: Sequential Predictive Modeling of Clinical Trial Outcome with Meta-Learning. arXiv preprint arXiv:2304.05352.

- **criteria_column** (*str*) – The column name of the criteria in the input data.
- **batch_size** (*int*) – Batch size for training.
- **n_trial_per_batch** (*int*) – Number of trials in each batch. If None, use all trials in the topic in each batch.
- **learning_rate** (*float*) – Learning rate for training.
- **weight_decay** (*float*) – Regularization strength for l2 norm; must be a positive float.
- **epochs** (*int*) – Number of training epochs.
- **evaluation_steps** (*int*) – Number of steps to evaluate the model.
- **warmup_ratio** (*float*) – Warmup ratio for learning rate scheduler.
- **device** (*str*) – Device to use for training and inference.
- **seed** (*int*) – Random seed.

Notes

add_trials(*seqtrain*, *df*)
Add new trials to the SequenceTrial object. It is used for the validation and test data.

Parameters

- **seqtrain** (*model_utils.data_structure.SequenceTrial*) – The sequence of clinical trials represented by SequenceTrial object.
- **df** (*pd.DataFrame*) – The dataframe of new trials that should be added to the input seqtrain.

fit(*train_data*, *valid_data=None*)
Train the SPOT model. It has two/three steps: 1. Encode the criteria using pretrained BERT to get the criteria embedding (optional). 2. Discover topics from the training data and transform the data into a SequenceTrial object. 3. Train the SPOT model.

Parameters

- **train_data** (*TrialOutcomeDataset*) – Training data, should be a *TrialOutcomeDataset* object.
- **valid_data** (*TrialOutcomeDataset*) – Validation data, should be a *TrialOutcomeDataset* object.

load_model(*input_dir=None*)
Load the model from a directory.

Parameters **input_dir** (*str or None*) – The directory to load the model.

predict(*data*, *target_trials=None*)
Predict the outcome of a clinical trial.

Parameters

- **data** (*TrialOutcomeDataset*) – A *TrialOutcomeDataset* object containing the data to predict.
- **target_trials** (*list*) – A list of trial ids to predict. If None, all trials in *data* will be predicted.

save_model(*output_dir=None*)
Save the model to a directory.

Parameters **output_dir** (*str or None*) – The directory to save the model. If None, use the default directory './checkpoints'.

transform_topic(*df*, *df_val=None*, *df_test=None*)
Transform the data into a SequenceTrial object.

Parameters

- **df** (*pandas.DataFrame*) – A dataframe containing the training data.
- **df_val** (*pandas.DataFrame*) – A dataframe containing the validation data.
- **df_test** (*pandas.DataFrame*) – A dataframe containing the test data.

CHAPTER

SEVEN

TASKS.TRIAL_PATIENT_MATCH

7.1 trial_patient_match.PatientTrialMatchBase

class pytrial.tasks.trial_patient_match.models.base.**PatientTrialMatchBase**(*experiment_id='test'*)

Abstract class for all sequential patient data simulations.

Parameters **experiment_id** (*str, optional (default = 'test')*) – The name of current experiment.

eval(*mode=False*)

Swith the model to the *validation* mode. Work samely as *model.eval()* in pytorch.

Parameters **mode** (*bool, optional (default = False)*) – If False, switch to the *validation* mode.

abstract fit(*train_data*)

Fit function needs to be implemented after subclass.

Parameters **train_data** (*Any*) – Training data.

abstract load_model(*checkpoint*)

Load the pretrained model from disk, needs to be implemented after subclass.

Parameters **checkpoint** (*str*) – The path to the checkpoint file.

abstract predict(*test_data*)

Prediction function needs to be implemented after subclass.

Parameters **test_data** (*Any*) – Testing data.

abstract save_model(*output_dir*)

Save the model to disk, needs to be implemented after subclass.

Parameters **output_dir** (*str*) – The path to the output directory.

train(*mode=True*)

Swith the model to the *training* mode. Work samely as *model.train()* in pytorch.

Parameters **mode** (*bool, optional (default = True)*) – If True, switch to the *training* mode.

7.2 trial_patient_match.DeepEnroll

class pytrial.tasks.trial_patient_match.models.deepenroll.**DeepEnroll**(*order*, *vocab_size*, *max_visit=20*, *word_dim=768*, *conv_dim=128*, *mem_dim=320*, *mlp_dim=512*, *demo_dim=3*, *margin=0.3*, *batch_size=512*, *epochs=50*, *learning_rate=0.001*, *weight_decay=0*, *num_worker=0*, *device='cpu'*, *experiment_id='test'*)

Bases: *pytrial.tasks.trial_patient_match.models.base.PatientTrialMatchBase*

Leverage DeepEnroll model for patient-trial matching[1].

One patient's label = [[0,1,2,3], [2,3,4,5]] where the first is a list of indices of inclusion criteria that the patient satisfies. the second is a list of indices of exclusion criteria that the patient does not satisfy. for the other criteria we dont know if the patient satisfied or not.

In this regard, we need to:

(1) predict "match" for all inclusion criteria of a trial for recruiting the patient

(2) predict "unmatch" for all exclusion criteria of a trial for recruiting the patient

Note that the dimension of the output logits indicates:

- 0 is unmatch
- 1 is match
- 2 is unknown

Parameters

- **order** (*list[str]*) – The order of events within each visit in the patient's EHR data, e.g., orders=['diag','med','prod'].
- **vocab_size** (*int*) – The vocabular size of each patient's EHR event types, e.g., diag, med, prod.
- **max_visit** (*int*) – Maximum number of visits to load for EHRs inputs.
- **word_dim** (*int*) – The dimension of input word embeddings for encoding eligibility criteria.
- **conv_dim** (*int*) – The dimension of convolutional layers for processing eligibility criteria embeddings.
- **mem_dim** (*int*) – The hidden dimension of the EHR memory network (encode patient EHRs).
- **mlp_dim** (*int*) – The hidden dimension of the MLP layers in the Query Network.

[1] Zhang, Xingyao, et al. "DeepEnroll: patient-trial matching with deep embedding and entailment prediction." Proceedings of The Web Conference 2020.

- **demo_dim** (*int*) – The input dimensions of patient demographic information, e.g., age, gender.
- **margin** (*float*) – The margin when compute nn.CosineEmbeddingLoss on patient embedding and inclusion/exclusion criteria embeddings. Refer to Eq. (12) of the reference paper.
- **epochs** (*int, optional (default=50)*) – Number of iterations (epochs) over the training data.
- **batch_size** (*int, optional (default=512)*) – Number of samples in each training batch.
- **learning_rate** (*float, optional (default=1e-3)*) – The learning rate.
- **weight_decay** (*float (default=0)*) – The weight decay during training.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – The model device.

Notes

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

Fit patient-trial matching model.

Parameters

- **train_data** (*dict*) – A dict contains patient and trial data.

  train_data =

  {

  'patient': *pytrial.tasks.trial_patient_match.data.PatientData*,

  'trial': *pytrial.tasks.trial_patient_match.data.TrialData*,

  }
- **valid_data** (*dict*) – A dict contains patient and trial data for evaluation. Same format as *train_data*.

load_model(*checkpoint*)

Load model and the pre-encoded trial embeddings from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model.

- If a directory, the only checkpoint file **.pth.tar* will be loaded.
- If a filepath, will load from this file.

predict(*test_data*, *return_dict=False*)

Predict the matching of patient and ECs of each target trial. Will make predictions for each patient, go through all included trials. For example, given 1000 patients and 10 trials, will generate predictions as the shape of `1000 x 10 x num_eligibility_criteria`.

Parameters

- **test_data** (*dict*) – A dict contains patient and trial data, respecitvely.

  test_data =

  {

  'patient': *pytrial.tasks.trial_patient_match.data.PatientData*,

'trial': *pytrial.tasks.trial_patient_match.data.TrialData*

}

- **return_dict** (*bool*) – If return results stored in dictionary, otherwise return tuples of results

save_model(*output_dir*)

Save the learned patient-match model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

7.3 trial_patient_match.COMPOSE

class pytrial.tasks.trial_patient_match.models.compose.**COMPOSE**(*order*, *vocab_size*, *max_visit=20*, *word_dim=768*, *conv_dim=128*, *mem_dim=320*, *mlp_dim=512*, *demo_dim=3*, *margin=0.3*, *batch_size=512*, *epochs=50*, *learning_rate=0.001*, *weight_decay=0*, *num_worker=0*, *device='cuda:0'*, *experiment_id='test'*)

Bases: *pytrial.tasks.trial_patient_match.models.base.PatientTrialMatchBase*

Leverage COMPOSE model for patient-trial matching[1].

One patient's label = [[0,1,2,3], [2,3,4,5]] where the first is a list of indices of inclusion criteria that the patient satisfies. the second is a list of indices of exclusion criteria that the patient does not satisfy. for the other criteria we dont know if the patient satisfied or not. In this regard, we need to: (1) predict "match" for all inclusion criteria of a trial for recruiting the patient (2) predict "unmatch" for all exclusion criteria of a trial for recruiting the patient

prediction logits # 0 is unmatch # 1 is match # 2 is unknown

Parameters

- **order** (*list[str]*) – The order of events within each visit in the patient's EHR data, e.g., orders=['diag','med','prod'].
- **vocab_size** (*int*) – The vocabular size of each patient's EHR event types, e.g., diag, med, prod.
- **max_visit** (*int*) – Maximum number of visits to load for EHRs inputs.
- **word_dim** (*int*) – The dimension of input word embeddings for encoding eligibility criteria.
- **conv_dim** (*int*) – The dimension of convolutional layers for processing eligibility criteria embeddings.
- **mem_dim** (*int*) – The hidden dimension of the EHR memory network (encode patient EHRs).
- **mlp_dim** (*int*) – The hidden dimension of the MLP layers in the Query Network.
- **demo_dim** (*int*) – The input dimensions of patient demographic information, e.g., age, gender.

[1] Gao, J., et al. (2020, August). COMPOSE: cross-modal pseudo-siamese network for patient trial matching. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 803-812).

- **margin** (*float*) – The margin when compute nn.CosineEmbeddingLoss on patient embedding and inclusion/exclusion criteria embeddings. Refer to Eq. (12) of the reference paper.
- **epochs** (*int, optional (default=50)*) – Number of iterations (epochs) over the training data.
- **batch_size** (*int, optional (default=512)*) – Number of samples in each training batch.
- **learning_rate** (*float, optional (default=1e-3)*) – The learning rate.
- **weight_decay** (*float (default=0)*) – The weight decay during training.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – The model device.

Notes

fit(*train_data*, *valid_data=None*)
Fit patient-trial matching model.

Parameters

- **train_data** (*dict*) – A dict contains patient and trial data, respectively.

  {

  'patient': *pytrial.tasks.trial_patient_match.data.PatientData*,

  'trial': *pytrial.tasks.trial_patient_match.data.TrialData*

  }
- **valid_data** (*dict*) – A dict contains patient and trial data for evaluation. Same format as *train_data*.

load_model(*checkpoint*)
Load model and the pre-encoded trial embeddings from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model. If a directory, the only checkpoint file **.pth.tar* will be loaded. If a filepath, will load from this file.

predict(*test_data*, *return_dict=False*)
Predict the matching of patient and ECs of each target trial. Will make predictions for each patient, go through all included trials. For example, given 1000 patients and 10 trials, will generate predictions as the shape of `1000 x 10 x num_eligibility_criteria`.

Parameters

- **test_data** (*dict*) – A dict contains patient and trial data, respecitvely.

  {

  'patient': *pytrial.tasks.trial_patient_match.data.PatientData*,

  'trial': *pytrial.tasks.trial_patient_match.data.TrialData*

  }
- **return_dict** (*bool*) – If return results stored in dictionary, otherwise return tuples of results

save_model(*output_dir*)
Save the learned patient-match model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

CHAPTER

EIGHT

TASKS.TRIAL_SEARCH

8.1 trial_search.TrialSearchBase

class pytrial.tasks.trial_search.models.base.**TrialSearchBase**(*experiment_id='test'*)

Bases: abc.ABC

Abstract class for all trial search algroithms.

Parameters **experiment_id** (*str, optional (default = 'test')*) – The name of current experiment.

abstract encode(*inputs*)

Encode input documents into embeddings.

Parameters **inputs** (*dict*) – The input documents.

abstract fit(*train_data*, *valid_data*)

Fit the model with training data. Need to implement in subclass.

Parameters

- **train_data** (*dict*) – Training data for model fitting.

  train_data = {

  'x': pd.DataFrame,

  'fields': list[str],

  'y': pd.Series or np.array,

  }

- **valid_data** (*dict*) – Validation data.

  valid_data = {

  'x': pd.DataFrame,

  'fields': list[str],

  'y': pd.Series or np.array,

  }

Returns **self** – The trained model.

Return type object

abstract load_model(*checkpoint*)

Parameters **checkpoint** (*str*) – The path to the saved model.

Returns **self** – The loaded pretrained model.

Return type object

abstract save_model(*output_dir*)

Parameters **output_dir** (*str*) – The directory to save the model states.

8.2 trial_search.BM25

8.3 trial_search.Doc2Vec

class pytrial.tasks.trial_search.models.doc2vec.**Doc2Vec**(*emb_dim=128*, *epochs=10*, *window=5*, *min_count=5*, *max_vocab_size=None*, *num_workers=4*, *experiment_id='test'*)

Bases: *pytrial.tasks.trial_search.models.base.TrialSearchBase*

Implement the Doc2Vec model for trial document similarity search.

Parameters

- **emb_dim** (*int, optional (default=128)*) – Dimensionality of the embedding vectors.
- **epochs** (*int, optional (default=10)*) – Number of iterations (epochs) over the corpus. Defaults to 10 for Doc2Vec.
- **window** (*int, optional (default=5)*) – The maximum distance between the current and predicted word within a sentence.
- **min_count** (*int, optional (default=5)*) – Ignores all words with total frequency lower than this.
- **max_vocab_size** (*int, optional*) – Limits the RAM during vocabulary building; if there are more unique words than this, then prune the infrequent ones. Every 10 million word types need about 1GB of RAM. Set to None for no limit.
- **num_workers** (*int, optional (default=4)*) – Use these many worker threads to train the model (=faster training with multicore machines).
- **experiment_id** (*str, optional (default='test')*) – The name of current experiment.

encode(*inputs*)

Encode input documents and output the document embeddings.

Parameters **inputs** (*dict*) – The documents which are to be encoded. *x*: a dataframe of trial documents. *fields*: the list of columns to be used in *x*.

inputs =

{

'x': pd.DataFrame,

'fields': list[str],

}

Returns **embs** – The encoded trial document embeddings.

Return type np.ndarray

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

Train the doc2vec model to get document embeddings for trial search.

Parameters **train_data** (*dict*) – Training corpus for the model.

- *x*: a dataframe of trial documents.
- *fields*: optional, the fields of documents to use for training. If not given, the model uses all fields in *x*.
- *tag*: optional, the field in *x* that serves as unique identifiers. Typically it is the *nct_id* of each trial. If not given, the model takes integer tags.

train_data =

{

'x': pd.DataFrame,

'fields': list[str],

'tag': str,

}

valid_data: Ignored. Not used, present here for API consistency by convention.

load_model(*checkpoint*)

Load the pretrained model from disk.

Parameters **checkpoint** (*str*) – The path to the pretrained model.

- If a directory, the only checkpoint file **.pth.tar* will be loaded.
- If a filepath, will load from this file.

predict(*test_data*, *top_k=10*)

Take the input document, find the most similar documents in the training corpus.

Parameters

- **test_data** (*dict*) – Trial docs to be predicted. *x*: a dataframe of trial documents. *fields*: optional, the fields of documents to use for training. If not given, the model uses all fields in *x*.

  test_data =

  {

  'x': pd.DataFrame,

  'fields': list[str],

  }

- **top_k** (*int, optional (default=10)*) – The number of top similar documents to be retrieved.

Returns **pred** – The sequence of *('key': similarities)* for the input test documents for each input trial.

Return type list[list[tuple[str, float]]]

save_model(*output_dir*)

Save the trained model.

Parameters **output_dir** (*str*) – The output directory to save. Checkpoint is saved with name *checkpoint.pth.tar* by default.

8.4 trial_search.WhitenBERT

class pytrial.tasks.trial_search.models.whiten_bert.**WhitenBERT**(*layer_mode='last_first'*, *bert_name='emilyalsentzer/Bio_ClinicalBERT'*, *device='cuda:0'*, *experiment_id='test'*)

Bases: *pytrial.tasks.trial_search.models.base.TrialSearchBase*

Implement a postprocessing method to improve BERT embeddings for similarity search[1].

Parameters

- **layer_mode** (*{'last_first', 'last"}*) – The mode of layer of embeddings to aggregate. 'last_first' means use the last layer and the first layer. 'last' means use the last layer only.
- **bert_name** (*str, optional (default = 'emilyalsentzer/Bio_ClinicalBERT')*) – The name of base BERT model used for encoding input texts.
- **device** (*str, optional (default = 'cuda:0')*) – The device of this model, typically be 'cpu' or 'cuda:0'.
- **experiment_id** (*str, optional (default = 'test')*) – The name of current experiment.

Notes

encode(*inputs*, *batch_size=None*, *num_workers=None*, *return_dict=True*, *verbose=True*)

Encode input documents and output the document embeddings.

Parameters

- **inputs** (*dict*) – The input documents to encode:
  - 'fields' is the list of fields to be encoded.
  - 'tag' is the unique index column name of each document, e.g., 'nctid'.

  inputs =

  {

  'x': pd.DataFrame,

  'fields': list[str],

  'tag': str,

  }
- **batch_size** (*int, optional*) – The batch size when encoding trials.
- **num_workers** (*int, optional*) – The number of workers when building the val dataloader.
- **return_dict** (*bool*) – Whether to return a dict of results.

[1] Huang, J., Tang, D., Zhong, W., Lu, S., Shou, L., Gong, M., … & Duan, N. (2021, November). WhiteningBERT: An Easy Unsupervised Sentence Embedding Approach. In Findings of the Association for Computational Linguistics: EMNLP 2021 (pp. 238-244).

- If set True, return dict[np.ndarray].
- Else, return np.ndarray with the order same as the input documents.

- **verbose** (*bool*) – Whether plot progress bar or not.

Returns **embs** – Encoded trial-level embeddings with key (tag) and value (embedding)..

Return type dict[np.ndarray]

evaluate(*test_data*)

Evaluate within the given trial and corresponding candidate trials.

Parameters **test_data** (*dict*) – The provided labeled dataset for test trials. Follow the format listed below.

test_data =

{

'x': pd.DataFrame,

'y': pd.DataFrame

}

Returns **results** – A dict of metrics and the values.

Return type dict[float]

Notes

x =

target_trial | trial1 | trial2 | trial3 |

nct01 | nct02 | nct03 | nct04 |

y =

label1 | label2 | label3 |

0 | 0 | 1 |

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

Go over all trials and encode them into embeddings. Note that this is a post-processing method based on a pretrained BERT model, so it does *NOT* need to be trained.

Parameters

- **train_data** (*dict*) – The data for encoding.
  - 'x' is the dataframe that contains multiple sections of a trial.
  - 'fields' is the list of fields to be encoded.

– 'tag' is the unique index column name of each document, e.g., 'nctid'.

train_data =

{

'x': pd.DataFrame,

'fields': list[str],

'tag': str,

}

- **valid_data** (*Not used.*) – This is a placeholder because this model does not need training.

load_model(*input_dir*)

Only load the embeddings. Do not load the model.

Parameters **input_dir** (*str*) – The input directory to load the model.

predict(*test_data*, *top_k=10*, *return_df=True*)

Predict the top-k relevant for input documents.

Parameters

- **test_data** (*dict*) – Share the same input format as the *train_data* in *fit* function. If *fields* and *tag* are not given, will reuse the ones used during training.

  test_data =

  {

  'x': pd.DataFrame,

  'fields': list[str],

  'tag': str,

  }

- **top_k** (*int*) – Number of retrieved candidates.
- **return_df** (*float*) –
  – If set True, return dataframe for the computed similarity ranking.
  – else, return rank_list=[[(doc1,sim1),(doc2,sim2)], [(doc1,sim1),...]].

Returns

- **rank** (*pd.DataFrame*) – A dataframe contains the top ranked NCT ids for each.
- **sim** (*pd.DataFrame*) – A dataframe contains the corresponding similarities.
- **rank_list** (*list[list[tuple]]*) – A list of tuples of top ranked docs and similarities.

save_model(*output_dir*)

Only save the embeddings. Do not save the model.

Parameters **output_dir** (*str*) – The output directory to save the model.

8.5 trial_search.Trial2Vec

class pytrial.tasks.trial_search.models.trial2vec.**Trial2Vec**(*fields=None*, *ctx_fields=None*, *tag_field='nct_id'*, *bert_name='emilyalsentzer/Bio_ClinicalBERT'*, *emb_dim=128*, *logit_scale_init_value=0.07*, *max_seq_length=128*, *epochs=10*, *batch_size=64*, *learning_rate=2e-05*, *weight_decay=0.0001*, *warmup_ratio=0*, *evaluation_steps=10*, *num_workers=0*, *device='cuda:0'*, *use_amp=False*, *experiment_id='test'*)

Bases: *pytrial.tasks.trial_search.models.base.TrialSearchBase*

Implement the Trial2Vec model for trial document similarity search[1].

Parameters

- **fields** (*list[str]*) – A list of fields of documents used as the *attribute* fields by Trial2Vec model.
- **ctx_fields** (*list[str]*) – A list of fields of documents used as the *context* fields by Trial2Vec model.
- **tag_field** (*str*) – The tag indicating trial documents, default to be 'nct_id'.
- **bert_name** (*str (default='emilyalsentzer/Bio_ClinicalBERT')*) – The base transformer-based encoder. Please find model names from the model hub of transformers (https://huggingface.co/models).
- **emb_dim** (*int, optional (default=768)*) – Dimensionality of the embedding vectors.
- **logit_scale_init_value** (*float, optional (default=0.07)*) – The logit scale or the temperature.
- **max_seq_length** (*int (default=128)*) – The maximum length of input tokens for the base encoder.
- **epochs** (*int, optional (default=10)*) – Number of iterations (epochs) over the corpus.
- **batch_size** (*int, optional (default=64)*) – Number of samples in each training batch.
- **learning_rate** (*float, optional (default=3e-5)*) – The learning rate.
- **weight_decay** (*float, optional (default=1e-4)*) – Weight decay applied for regularization.
- **warmup_ratio** (*float (default=0)*) – How many steps used for warmup training. If set 0, not warmup.
- **evaluation_steps** (*int (default=10)*) – How many iterations while we print the training loss and conduct evaluation if evaluator is given.

[1] Wang, Z., & Sun, J. (2022). Trial2Vec: Zero-Shot Clinical Trial Document Similarity Search using Self-Supervision. Findings of EMNLP 2022.

- **num_workers** (*int, optional (default=0)*) – Use these many worker threads to train the model (=faster training with multicore machines).
- **device** (*str or torch.device (default='cuda:0')*) – The device to put the model on.
- **use_amp** (*bool (default=False)*) – Whether or not use mixed precision training.
- **experiment_id** (*str, optional (default='test')*) – The name of current experiment.

Notes

encode(*inputs*, *batch_size=None*, *num_workers=None*, *return_dict=True*, *verbose=True*)

Encode input documents and output the document embeddings.

Parameters

- **inputs** (*dict*) – inputs = {

  'x': pd.DataFrame,

  'fields': list[str],

  'ctx_fields': list[str],

  'tag': str,

  }

  Share the same input format as the *train_data* in *fit* function. If *fields*, *ctx_fields*, *tag* are not given, will reuse the ones used during training.
- **batch_size** (*int, optional*) – The batch size when encoding trials.
- **num_workers** (*int, optional*) – The number of workers when building the val dataloader.
- **return_dict** (*bool*) – If set True, return dict[np.ndarray]. Else, return np.ndarray with the order same as the input documents.
- **verbose** (*bool*) – Whether plot progress bar or not.

Returns **embs** – Encoded trial-level embeddings with key (tag) and value (embedding)..

Return type dict[np.ndarray]

evaluate(*test_data*)

Evaluate within the given trial and corresponding candidate trials.

Parameters **test_data** (*dict*) – test_data =

{

'x': pd.DataFrame,

'y': pd.DataFrame

}

The provided labeled dataset for test trials. Follow the format listed above.

Returns **results** – A dict of metrics and the values.

Return type dict[float]

Notes

x =

target_trial | trial1 | trial2 | trial3 |

nct01 | nct02 | nct03 | nct04 |

y =

label1 | label2 | label3 |

0 | 0 | 1 |

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

Train the trial2vec model to get document embeddings for trial search.

Parameters

- **train_data** (*dict*) – train_data =

  {

  'x': pd.DataFrame,

  'fields': list[str],

  'ctx_fields': list[str],

  'tag': str,

  }

  Training corpus for the model.

  - *x*: a dataframe of trial documents.
  - *fields*: optional, the fields of documents to use for training as key attributes. If not given, the model uses all fields in *x*.
  - *ctx_fields*: optional, the fields of documents which belong to context components. If not given, the model will only learn from *fields*.
  - *tag*: optional, the field in *x* that serves as unique identifiers. Typically it is the *nct_id* of each trial. If not given, the model takes integer tags.

- **valid_data** (*dict={'x':pd.DataFrame 'y':np.ndarray}.*) – Validation data used for identifying the best checkpoint during the training. Need to rewrite the function: *get_val_dataloader*.

from_pretrained(*input_dir=None*)

Download pretrained Trial2Vec model.

load_model(*checkpoint*)

Load model and the pre-encoded trial embeddings from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model. If a directory, the only checkpoint file **.pth.tar* will be loaded. If a filepath, will load from this file.

predict(*test_data*, *top_k=10*, *return_df=True*, *skip_pretrained=False*)

Predict the top-k relevant for input documents.

Parameters

- **test_data** (*dict*) – test_data =

  {

  'x': pd.DataFrame,

  'fields': list[str],

  'ctx_fields': list[str],

  'tag': str,

  }

  Share the same input format as the *train_data* in *fit* function. If *fields*, *ctx_fields*, *tag* are not given, will reuse the ones used during training.

- **top_k** (*int*) – Number of retrieved candidates.
- **return_df** (*float*) – Whether or not return dataframe for the computed similarity ranking.
  - If set True, return (rank, sim);
  - else, return rank_list=[[(doc1,sim1),(doc2,sim2)], [(doc1,sim1),...]].
- **skip_pretrained** (*bool*) – Whether or not skip encoding the trial which has been in the self.trial_embs. If set True, will skip encoding the trial, and get the trial embeddings by lookup from self.trial_embs.

Returns

- **rank** (*pd.DataFrame*) – A dataframe contains the top ranked NCT ids for each.
- **sim** (*pd.DataFrame*) – A dataframe contains the corresponding similarities.
- **rank_list** (*list[list[tuple]]*) – A list of tuples of top ranked docs and similarities.

save_model(*output_dir*)

Parameters **output_dir** (*str*) – The directory to save the model states.

update_emb(*emb_dict*)

Update trial embeds: add or modify.

Parameters **emb_dict** (*dict[np.ndarray]*) – The tag and corresponding trial embeddings to updated.

CHAPTER

NINE

TASKS.TRIAL_SIMULATION

9.1 tasks.trial_simulation.tabular

9.1.1 trial_simulation.tabular.TabularSimulationBase

class pytrial.tasks.trial_simulation.tabular.base.**TabularSimulationBase**(*experiment_id='trial_simulation.tabular'*)

Bases: abc.ABC

Abstract class for all tabular simulations based on tabular patient data.

Parameters **experiment_id** (*str, optional (default = 'test')*) – The name of current experiment.

abstract fit(*train_data*)

Fit the model to the given training data. Need to be implemented by subclass.

Parameters **train_data** (*Any*) – The training data.

abstract load_model(*checkpoint*)

Load model from the given directory. Need to be implemented by subclass.

Parameters **checkpoint** (*str*) – The given filepath (e.g., ./checkpoint/model.pth)

abstract predict(*number_of_predictions*)

Generate synthetic data after the model trained. Need to be implemented by subclass.

Parameters **number_of_predictions** (*Any*) – Number of synthetic data to be generated.

abstract save_model(*output_dir*)

Save the model to the given directory. Need to be implemented by subclass.

Parameters **output_dir** (*str*) – The given directory to save the model.

9.1.2 trial_simulation.tabular.GaussianCopula

class pytrial.tasks.trial_simulation.tabular.gaussian_copula.**GaussianCopula**(*experiment_id='trial_simulation.tabular.gaussiancopula'*)

Bases: *pytrial.tasks.trial_simulation.tabular.base.TabularSimulationBase*

Implement Gaussian Copula model for tabular patient simulation.

Parameters **experiment_id** (*str, optional (default='trial_simulation.tabular.gaussiancopula')*) – The name of current experiment. Decide the saved model checkpoint name.

fit(*train_data*)

Train gaussian copula model to simulate patient outcome with tabular input data.

Parameters **train_data** (*dict or TabularPatientBase*) – The training data, which is the real tabular patient data.

load_model(*checkpoint=None*)
Save the learned gaussian copula model to the disk.

Parameters **checkpoint** (*str or None*) – The path to the saved model.

- If a directory, the only checkpoint file *.model* will be loaded.
- If a filepath, will load from this file;
- If None, will load from *self.checkout_dir*.

predict(*n=200*)
simulate a new tabular data with number_of_predictions.

Parameters **n** (*int*) – The number of synthetic samples to generation.

Returns **ypred** – A new tabular data simulated by the model

Return type *TabularPatientBase*

save_model(*output_dir=None*)
Save the learned gaussian copula model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

9.1.3 trial_simulation.tabular.CopulaGAN

class pytrial.tasks.trial_simulation.tabular.copula_gan.**CopulaGAN**(*embedding_dim=128*, *generator_dim=(256, 256)*, *discriminator_dim=(256, 256)*, *generator_lr=0.0002*, *generator_decay=1e-06*, *discriminator_lr=0.0002*, *discriminator_decay=1e-06*, *batch_size=500*, *discriminator_steps=1*, *log_frequency=True*, *verbose=True*, *epochs=50*, *pac=10*, *cuda=False*, *experiment_id='trial_simulation.tabular.copulagan'*)

Bases: *pytrial.tasks.trial_simulation.tabular.base.TabularSimulationBase*

Implement CopulaGAN model for tabular patient data simulation[1].

Parameters

- **embedding_dim** (*int*) – Size of the random sample passed to the Generator. Defaults to 128.
- **generator_dim** (*tuple or list of int:*) – Size of the output samples for each one of the Residuals. A Residual Layer will be created for each one of the values provided. Defaults to (256, 256).

[1] Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional gan. Advances in Neural Information Processing Systems, 32.

- **discriminator_dim** (*tuple or list of ints*) – Size of the output samples for each one of the Discriminator Layers. A Linear Layer will be created for each one of the values provided. Defaults to (256, 256).
- **generator_lr** (*float*) – Learning rate for the generator. Defaults to 2e-4.
- **generator_decay** (*float*) – Generator weight decay for the Adam Optimizer. Defaults to 1e-6.
- **discriminator_lr** (*float*) – Learning rate for the discriminator. Defaults to 2e-4.
- **discriminator_decay** (*float*) – Discriminator weight decay for the Adam Optimizer. Defaults to 1e-6.
- **batch_size** (*int*) – Number of data samples to process in each step.
- **discriminator_steps** (*int*) – Number of discriminator updates to do for each generator update. From the WGAN paper: https://arxiv.org/abs/1701.07875. WGAN paper default is 5. Default used is 1 to match original CTGAN implementation.
- **log_frequency** (*boolean*) – Whether to use log frequency of categorical levels in conditional sampling. Defaults to `True`.
- **verbose** (*boolean*) – Whether to have print statements for progress results. Defaults to `True`.
- **epochs** (*int*) – Number of training epochs. Defaults to 10.
- **pac** (*int*) – Number of samples to group together when applying the discriminator. Defaults to 10.
- **cuda** (*bool or str*) –
  - If `True`, use CUDA. If a `str`, use the indicated device.
  - If `False`, do not use cuda at all.
- **experiment_id** (*str, optional*) – The name of current experiment. Decide the saved model checkpoint name.

Notes

fit(*train_data*)

Train CopulaGAN model to simulate patient outcome with tabular input data.

Parameters **train_data** (`TabularPatientBase`) – The training data for the model.

load_model(*checkpoint=None*)

Save the learned CopulaGAN model to the disk.

Parameters **checkpoint** (*str or None*) – The path to the saved model. - If a directory, the only checkpoint file *.model* will be loaded. - If a filepath, will load from this file; - If None, will load from *self.checkout_dir*.

predict(*n=200*)

Simulate new tabular data with number_of_predictions.

Parameters **n** (*int*) – The number of synthetic data to generate.

Returns **ypred** – A new tabular data simulated by the model

Return type *TabularPatientBase*

save_model(*output_dir=None*)
Save the learned CopulaGAN model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

9.1.4 trial_simulation.tabular.TVAE

class pytrial.tasks.trial_simulation.tabular.tvae.**TVAE**(*embedding_dim=128*, *compress_dims=(128, 128)*, *decompress_dims=(128, 128)*, *l2scale=1e-05*, *batch_size=500*, *epochs=50*, *loss_factor=2*, *cuda=False*, *experiment_id='trial_simulation.tabular.tvae'*)
Bases: *pytrial.tasks.trial_simulation.tabular.base.TabularSimulationBase*

Implement TVAE model for tabular patient data simulation[1].

Parameters

- **embedding_dim** (*int*) – Size of the random sample passed to the Generator. Defaults to 128.
- **compress_dims** (*tuple or list[int]*) – Size of each hidden layer in the encoder. Defaults to (128, 128).
- **decompress_dims** (*tuple or list[int]*) – Size of each hidden layer in the decoder. Defaults to (128, 128).
- **l2scale** (*int*) – Regularization term. Defaults to 1e-5.
- **batch_size** (*int*) – Number of data samples to process in each step.
- **epochs** (*int*) – Number of training epochs. Defaults to 300.
- **loss_factor** (*int*) – Multiplier for the reconstruction error. Defaults to 2.
- **cuda** (*bool or str*) –
  - If `True`, use CUDA. If a `str`, use the indicated device.
  - If `False`, do not use cuda at all.
- **experiment_id** (*str, optional*) – The name of current experiment. Decide the saved model checkpoint name.

Notes

fit(*train_data*)
Train TVAE model to simulate patient data with tabular input data.

Parameters **train_data** (TabularPatientBase) – The training data for TVAE model.

load_model(*checkpoint=None*)
Load the learned TVAE model from the disk.

Parameters **checkpoint** (*str or None*) – The path to the checkpoint file.

- If a directory, the only checkpoint file *.model* will be loaded.
- If a filepath, will load from this file;

[1] Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional gan. Advances in Neural Information Processing Systems, 32.

- If None, will load from *self.checkout_dir*.

predict(*n=200*)
simulate a new tabular data with n.

Parameters **n** (*int*) – The number of new data to simulate.

Returns **ypred** – A new tabular data simulated by the model

Return type TanularPatientBase

save_model(*output_dir=None*)
Save the learned TVAE model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

9.1.5 trial_simulation.tabular.CTGAN

class pytrial.tasks.trial_simulation.tabular.ct_gan.**CTGAN**(*embedding_dim=128*, *generator_dim=(256, 256)*, *discriminator_dim=(256, 256)*, *generator_lr=0.0002*, *generator_decay=1e-06*, *discriminator_lr=0.0002*, *discriminator_decay=1e-06*, *batch_size=500*, *discriminator_steps=1*, *log_frequency=True*, *verbose=True*, *epochs=50*, *pac=10*, *cuda=False*, *experiment_id='trial_simulation.tabular.ctgan'*)

Bases: *pytrial.tasks.trial_simulation.tabular.base.TabularSimulationBase*

Implement CTGAN model for patient level tabular data generation[1].

Parameters

- **embedding_dim** (*int*) – Size of the random sample passed to the Generator. Defaults to 128.
- **generator_dim** (*tuple or list of ints*) – Size of the output samples for each one of the Residuals. A Residual Layer will be created for each one of the values provided. Defaults to (256, 256).
- **discriminator_dim** (*tuple or list of ints*) – Size of the output samples for each one of the Discriminator Layers. A Linear Layer will be created for each one of the values provided. Defaults to (256, 256).
- **generator_lr** (*float*) – Learning rate for the generator. Defaults to 2e-4.
- **generator_decay** (*float*) – Generator weight decay for the Adam Optimizer. Defaults to 1e-6.
- **discriminator_lr** (*float*) – Learning rate for the discriminator. Defaults to 2e-4.
- **discriminator_decay** (*float*) – Discriminator weight decay for the Adam Optimizer. Defaults to 1e-6.
- **batch_size** (*int*) – Number of data samples to process in each step.

[1] Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling tabular data using conditional gan. Advances in Neural Information Processing Systems, 32.

- **discriminator_steps** (*int*) – Number of discriminator updates to do for each generator update. From the WGAN paper: https://arxiv.org/abs/1701.07875. WGAN paper default is 5. Default used is 1 to match original CTGAN implementation.
- **log_frequency** (*bool*) – Whether to use log frequency of categorical levels in conditional sampling. Defaults to `True`.
- **verbose** (*bool*) – Whether to have print statements for progress results. Defaults to `True`.
- **epochs** (*int*) – Number of training epochs. Defaults to 300.
- **pac** (*int*) – Number of samples to group together when applying the discriminator. Defaults to 10.
- **cuda** (*bool or str*) – If `True`, use CUDA. If a `str`, use the indicated device. If `False`, do not use cuda at all.
- **experiment_id** (*str, optional (default='trial_simulation.tabular.ctgan')*) – The name of current experiment. Decide the saved model checkpoint name.

Notes

fit(*train_data*)
Train CTGAN model to simulate tabular patient data.

Parameters **train_data** (`TabularPatientBase`) – The training data.

load_model(*checkpoint=None*)
Save the learned CTGAN model to the disk.

Parameters **checkpoint** (*str or None*) – If a directory, the only checkpoint file *.model* will be loaded. If a filepath, will load from this file; If None, will load from *self.checkout_dir*.

predict(*n=200*)
simulate a new tabular data with number_of_predictions.

Parameters **n** (*int*) – number of synthetic records going to generate.

Returns **ypred** – A new tabular data simulated by the model

Return type dataset, same as the input dataset

save_model(*output_dir=None*)
Save the learned CTGAN model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

9.1.6 trial_simulation.tabular.MedGAN

class pytrial.tasks.trial_simulation.tabular.med_gan.**MedGAN**(*embedding_dim=128*, *random_dim=128*, *generator_dims=(128, 128)*, *discriminator_dims=(256, 128, 1)*, *compress_dims=()*, *decompress_dims=()*, *bn_decay=0.99*, *l2scale=0.001*, *pretrain_epoch=200*, *batch_size=1000*, *epochs=2000*, *device='cpu'*, *experiment_id='trial_simulation.tabular.medgan'*, *verbose=False*)

Bases: *pytrial.tasks.trial_simulation.tabular.base.TabularSimulationBase*

Implement MedGAN model for patient level tabular data generation[1].

Parameters

- **embedding_dim** (*int, default 128*) – Dimension of embedding layer.
- **random_dim** (*int, default 128*) – Dimension of random noise.
- **generator_dims** (*tuple, default (128, 128)*) – Dimension of generator layers.
- **discriminator_dims** (*tuple, default (256, 128, 1)*) – Dimension of discriminator layers.
- **compress_dims** (*tuple, default ()*) – Dimension of compressed embedding layer. datadim -> embedding_dim
- **decompress_dims** (*tuple, default ()*) – Dimension of decompressed embedding layer. embedding_dim -> datadim
- **bn_decay** (*float, default 0.99*) – Decay rate of batch normalization.
- **l2scale** (*float, default 0.001*) – L2 regularization scale.
- **pretrain_epoch** (*int, default 200*) – Number of pretrain epochs.
- **batch_size** (*int, default 1000*) – Batch size for training.
- **epochs** (*int, default 1000*) – Number of epochs for training.
- **experiment_id** (*str*) – Experiment id for logging.
- **verbose** (*bool*) – Whether to print training information.

Notes

fit(*train_data*)
Train MedGAN model to generate synthetic tabular patient data.

Parameters **train_data** (TabularPatientBase) – Training data.

load_model(*checkpoint*)
Load model from checkpoint.

[1] Choi, E., Biswal, S., Malin, B., Duke, J., Stewart, W. F., & Sun, J. (2017, November). Generating multi-label discrete patient records using generative adversarial networks. In Machine learning for healthcare conference (pp. 286-305). PMLR.

Parameters **checkpoint** (*str*) – Path to checkpoint. If a directory is given, will load the latest checkpoint in the directory. If a filepath is given, will load the checkpoint from the filepath. If set None, will load from default directory *self.checkpoint_dir*.

predict(*n*)
Generate synthetic tabular patient data.

Parameters **n** (*int*) – Number of samples to generate.

Returns **data** – Generated synthetic data.

Return type np.ndarray

save_model(*output_dir*)
Save model to checkpoint.

Parameters **output_dir** (*str*) – Output directory. If set None, will save to default directory *self.checkpoint_dir*.

9.2 tasks.trial_simulation.sequence

9.2.1 trial_simulation.sequence.SequenceSimulationBase

9.2.2 trial_simulation.sequence.RNNGAN

class pytrial.tasks.trial_simulation.sequence.rnn_gan.**RNNGAN**(*vocab_size*, *order*, *max_visit=20*, *emb_size=64*, *n_rnn_layer=2*, *rnn_type='lstm'*, *bidirectional=False*, *padding_idx=None*, *learning_rate=0.0001*, *weight_decay=0.0001*, *batch_size=64*, *epochs=10*, *num_worker=0*, *device='cuda:0'*, *experiment_id='trial_simulation.sequence.rnn_gan'*)

Bases: pytrial.tasks.trial_simulation.sequence.base.SequenceSimulationBase

Implement an RNN based GAN model for longitudinal patient records simulation. The GAN part was proposed by Choi et al.[1].

Parameters

- **vocab_size** (*list[int]*) – A list of vocabulary size for different types of events, e.g., for diagnosis, procedure, medication.
- **order** (*list[str]*) – The order of event types in each visits, e.g., ['diag', 'prod', 'med']. Visit = [diag_events, prod_events, med_events], each event is a list of codes.
- **max_visit** (*int*) – The maximum number of visits for input event codes.
- **emb_size** (*int*) – Embedding size for encoding input event codes.
- **n_rnn_layer** (*int*) – Number of RNN layers for encoding historical events.
- **rnn_type** (*str*) – Pick RNN types in ['rnn','lstm','gru']

[1] Choi, E., et al. (2017, November). Generating multi-label discrete patient records using generative adversarial networks. In ML4HC (pp. 286-305). PMLR.

- **bidirectional** (*bool*) – If True, it encodes historical events in bi-directional manner.
- **padding_idx** (*int(default=None)*) – Set the padding index for input events embedding. If set None, then no padding index will be specified.
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **weigth_decay** (*float*) – Regularization strength for l2 norm; must be a positive float. Smaller values specify weaker regularization.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.
- **device** (*str*) – The model device.

Notes

fit(*train_data*)
Train model with sequential patient records.

Parameters **train_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events.

load_model(*checkpoint*)
Load model and the pre-encoded trial embeddings from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model.

- If a directory, the only checkpoint file **.pth.tar* will be loaded.
- If a filepath, will load from this file.

predict(*test_data*, *n=None*, *n_per_sample=None*, *return_tensor=True*)
Generate synthetic records based on input real patient seq data.

Parameters

- **test_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events.
- **n** (*int*) – How many samples in total will be generated.
- **n_per_sample** (*int*) – How many samples generated based on each indivudals.
- **return_tensor** (*bool*) – If *True*, return output generated records in tensor format (n, n_visit, n_event), good for later predictive modeling. If *False, return records in `SequencePatient* format.

save_model(*output_dir*)
Save the learned simulation model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

9.2.3 trial_simulation.sequence.EVA

class pytrial.tasks.trial_simulation.sequence.eva.**EVA**(*vocab_size*, *order*, *max_visit=20*, *emb_size=64*, *latent_dim=32*, *n_rnn_layer=2*, *learning_rate=0.001*, *batch_size=64*, *epochs=20*, *num_worker=0*, *device='cpu'*, *experiment_id='trial_simulation.sequence.eva'*)

Bases: pytrial.tasks.trial_simulation.sequence.base.SequenceSimulationBase

Implement a VAE based model for longitudinal patient records simulation[1].

Parameters

- **vocab_size** (*list[int]*) – A list of vocabulary size for different types of events, e.g., for diagnosis, procedure, medication.
- **order** (*list[str]*) – The order of event types in each visits, e.g., ['diag', 'prod', 'med']. Visit = [diag_events, prod_events, med_events], each event is a list of codes.
- **max_visit** (*int*) – Maximum number of visits.
- **emb_size** (*int*) – Embedding size for encoding input event codes.
- **latent_dim** (*int*) – Size of final latent dimension between the encoder and decoder
- **n_rnn_layer** (*int*) – Number of RNN layers for encoding historical events.
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.
- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.

Notes

fit(*train_data*)

Train model with sequential patient records.

Parameters **train_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events.

load_model(*checkpoint*)

Load model and the pre-encoded trial embeddings from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model.

- If a directory, the only checkpoint file **.pth.tar* will be loaded.
- If a filepath, will load from this file.

predict(*n*, *return_tensor=False*)

Generate synthetic records

Parameters

- **n** (*int*) – How many samples in total will be generated.
- **return_tensor** (*bool*) –

[1] Biswal, S., et al. (2020, December). EVA: Generating Longitudinal Electronic Health Records Using Conditional Variational Autoencoders.

- If *True*, return output generated records in tensor format (n, n_visit, n_event), good for later predictive modeling.
- If *False, return records in `SequencePatient* format.

save_model(*output_dir*)

Save the learned simulation model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

9.2.4 trial_simulation.sequence.SynTEG

class pytrial.tasks.trial_simulation.sequence.synteg.**SynTEG**(*vocab_size*, *order*, *max_visit=20*, *max_code_per_visit=20*, *emb_size=64*, *hidden_dim=32*, *condition_dim=32*, *n_head=4*, *n_rnn_layer=2*, *z_dim=64*, *g_dims=[32, 32, 64, 64]*, *d_dims=[64, 32, 32]*, *learning_rate=0.001*, *batch_size=64*, *epochs=20*, *num_worker=0*, *device='cpu'*, *experiment_id='trial_simulation.sequence.synteg'*)

Bases: pytrial.tasks.trial_simulation.sequence.base.SequenceSimulationBase

Implement a GAN based model for longitudinal patient records simulation[1].

Parameters

- **vocab_size** (*list[int]*) – A list of vocabulary size for different types of events, e.g., for diagnosis, procedure, medication.
- **order** (*list[str]*) – The order of event types in each visits, e.g., ['diag', 'prod', 'med']. Visit = [diag_events, prod_events, med_events], each event is a list of codes.
- **max_visit** (*int*) – Maximum number of visits.
- **max_code_per_visit** (*int*) – Maximum number of medical codes in a single visit.
- **emb_size** (*int*) – Embedding size for encoding input event codes.
- **hidden_dim** (*int*) – Size of intermediate hidden dimension for RNN and Feed Forward layers
- **condition_dim** (*int*) – Size of intermediate dimension for encoding medical history to condition the GAN
- **n_head** (*int*) – Number of attention heads
- **n_rnn_layer** (*int*) – Number of RNN layers for encoding historical events.
- **z_dim** (*int*) – Dimension of noise vector passed into the GAN Generator
- **g_dims** (*list*) – List of ints for intermediate GAN Generator dimensionalities
- **d_dims** (*list*) – List of ints for intermediate GAN Discriminator dimensionalities
- **learning_rate** (*float*) – Learning rate for optimization based on SGD. Use torch.optim.Adam by default.

[1] Zhang, Ziqi, et al. (2021, March). SynTEG: a framework for temporal structured electronic health data simulation. Journal of the American Medical Informatics Association 28.3.

- **batch_size** (*int*) – Batch size when doing SGD optimization.
- **epochs** (*int*) – Maximum number of iterations taken for the solvers to converge.
- **num_worker** (*int*) – Number of workers used to do dataloading during training.

Notes

fit(*train_data*)
Train model with sequential patient records.

Parameters **train_data** (SequencePatientBase) – A *SequencePatientBase* contains patient records where 'v' corresponds to visit sequence of different events.

load_model(*checkpoint*)
Load model and the pre-encoded trial embeddings from the given checkpoint dir.

Parameters **checkpoint** (*str*) – The input dir that stores the pretrained model. - If a directory, the only checkpoint file **.pth.tar* will be loaded. - If a filepath, will load from this file.

predict(*n*, *return_tensor=True*)
Generate synthetic records

Parameters

- **n** (*int*) – How many samples in total will be generated.
- **return_tensor** (*bool*) – If *True*, return output generated records in tensor format (n, n_visit, n_event), good for later predictive modeling. If *False, return records in `SequencePatient* format.

save_model(*output_dir*)
Save the learned simulation model to the disk.

Parameters **output_dir** (*str or None*) – The dir to save the learned model. If set None, will save model to *self.checkout_dir*.

9.2.5 trial_simulation.sequence.PromptEHR

9.2.6 trial_simulation.sequence.KNNSampler

9.2.7 trial_simulation.sequence.TWIN

9.2.8 trial_simulation.sequence.evaluation

CHAPTER

TEN

MODEL_UTILS

10.1 model_utils.bert

class pytrial.model_utils.bert.**BERT**(*bertname='emilyalsentzer/Bio_ClinicalBERT'*, *proj_dim=None*, *max_length=512*, *device='cpu'*)

Bases: torch.nn.modules.module.Module

The pretrained BERT model for getting text embeddings.

Parameters

- **bertname** (*str (default='emilyalsentzer/Bio_ClinicalBERT')*) – The name of pretrained bert to get from huggingface models hub: https://huggingface.co/models. Or pass the dir where the local pretrained bert is available.
- **proj_dim** (*int or None*) – A linear projection head added on top of the bert encoder. Note that if given, the projection head is RANDOMLY initialized and needs further training.
- **max_length** (*int*) – Maximum acceptable number of tokens for each sentence.
- **device** (*str*) – The device of this model, typically be 'cpu' or 'cuda:0'.

Examples

```
>>> model = BERT()
>>> emb = model.encode('The goal of life is comfort.')
>>> print(emb.shape)
```

encode(*input_text*, *is_train=False*, *batch_size=None*)

Encode the input texts into embeddings.

Parameters

- **input_text** (*str or list[str]*) – A sentence or a list of sentences to be encoded.
- **is_train** (*bool*) – Set True if this model's parameters will update by learning.
- **batch_size** (*int*) – How large batch size to use when encoding long documents with many sentences. When set *None*, will encode all sentences at once.

Returns **outputs** – The encoded sentence embeddings with size [num_sent, emb_dim]

Return type torch.Tensor

forward(*input_ids*, *attention_mask=None*, *token_type_ids=None*, *return_hidden_states=False*)

Forward pass of the model.

Parameters

- **input_ids** (*torch.Tensor*) – The input token ids with shape [batch_size, seq_len].
- **attention_mask** (*torch.Tensor*) – The attention mask with shape [batch_size, seq_len].
- **token_type_ids** (*torch.Tensor*) – The token type ids with shape [batch_size, seq_len].
- **return_hidden_states** (*bool*) – Whether to return the hidden states of all layers.

10.2 model_utils.icd

class pytrial.model_utils.icd.**ICD9Graph**(*input_dir=None*)
Bases: pytrial.model_utils.icd.ICDGraphBase

Get an ICD-9 knowledge graph to query parental and children nodes for each code.

Returns

- **self.graph** (*nx.DiGraph*) – The hierarchy of ICD codes stored as graph in networkx.
- **self.codes** (*list[str]*) – All the unique codes.

class pytrial.model_utils.icd.**ICD10Graph**(*input_dir=None*, *version='2021'*)
Bases: pytrial.model_utils.icd.ICDGraphBase

Get an ICD-10 knowledge graph to query parental and children nodes for each code.

Parameters

- **input_dir** (*str*) – The dir that stores the hierarchy files.
- **version** (*{'2022', '2021','2020','2019'}*) – The version of ICD-10 codes.

Returns

- **self.graph** (*nx.DiGraph*) – The hierarchy of ICD codes stored as graph in networkx.
- **self.codes** (*list[str]*) – All the unique codes.

class pytrial.model_utils.icd.**ICD9_DX_VOC**(*input_dir='./resources'*)
Bases: object

Get a vocabulary containing the mapping of ICD9 Diagnosis code and its names.

Parameters **input_dir** (*str*) – The dir that stores ICD9-dx file.

code2desc(*code*)
Get description of codes.

Parameters **code** (*str or List[str]*) – The input icd code or list of codes.

class pytrial.model_utils.icd.**ICD9_SG_VOC**(*input_dir='./resources'*)
Bases: object

Get a vocabulary containing the mapping of ICD9 procedure code and its names.

Parameters **input_dir** (*str*) – The dir that stores ICD9-sg file.

code2desc(*code*)
Get description of codes.

Parameters **code** (*str or List[str]*) – The input icd code or list of codes.

pytrial.model_utils.icd.**get_icd10_from_nih**(*term*)
Query related ICD-10 codes for input terms.

Parameters **term** (*str or list[str]*) – Disease names or a list of disease names.

Returns **Outputs ICD codes**

Return type list[str] or list[list[str]]

pytrial.model_utils.icd.**get_icd9dx_from_nih**(*term*)
Query related ICD-9-CM diagnosis codes for input terms.

Parameters **term** (*str or list[str]*) – Disease names or a list of disease names.

Returns **Outputs ICD codes**

Return type list[str] or list[list[str]]

pytrial.model_utils.icd.**get_icd9sg_from_nih**(*term*)
Query related ICD-9-CM procedure codes for input terms.

Parameters **term** (*str or list[str]*) – Disease names or a list of disease names.

Returns **Outputs ICD codes**

Return type list[str] or list[list[str]]

pytrial.model_utils.icd.**get_condition_synonym_from_nih**(*term*)
Query relevant medical conditions taking input symptoms/diseases using API: https://clinicaltables.nlm.nih.gov/apidoc/conditions/v3/doc.html

Parameters **term** (*str or list[str]*) – Disease names or a list of disease names.

Returns **Outputs ICD codes**

Return type list[str] or list[list[str]]

10.3 model_utils.drug

class pytrial.model_utils.drug.**DrugTransformer**
Bases: object

Provide a series of drug-related functions for

(1) drug name to atc / atc to drug name

(2) drug name to ndc-11 / ndc-11 to drug name

(3) drug name to smiles (molecule structure)

(4) atc to smiles

(5) ndc-11 to smiles

(6) ndc-11 to atc / atc to ndc

To convert ndc-10 ditis to ndc-11, use *convert_ndc10_ndc11*.

atc2name(*code*)

Parameters **code** (*str or list[str]*) – ATC4 codes.

Returns **names** – Drug names.

Return type list[str] or list[list[str,None]]

atc2ndc(*code*)

Parameters **code** (*str or list[str]*) – ATC-4 codes.

Returns **NDC-11 codes**

Return type list[str] or list[list[str,None]]

atc2smiles(*code*)

Parameters **code** (*str or list[str]*) – ATC4 codes.

Returns **smiles** – Drug molecule structures represented by SMILES.

Return type list[str] or list[list[str,None]]

name2atc(*name*)

Parameters **name** (*str or list[str]*) – Drug names.

Returns **codes** – ATC4 codes.

Return type list[str] or list[list[str,None]]

name2ndc(*name*)

Parameters **name** (*str or list[str]*) – Drug names.

Returns **ndc codes**

Return type list[str] or list[list[str,None]]

name2smiles(*name*)

Parameters **name** (*str or list[str]*) – Drug names.

Returns **smiles** – Drug molecule structures represented by SMILES.

Return type list[str] or list[list[str,None]]

ndc2atc(*code*)

Parameters **code** (*str or list[str]*) – NDC-11 codes.

Returns **atc codes**

Return type list[str] or list[list[str,None]]

ndc2name(*code*)

Parameters **code** (*str or list[str]*) – NDC-11 codes.

Returns **name** – Drug names.

Return type list[str] or list[list[str,None]]

ndc2smiles(*code*)

Parameters **name** (*str or list[str]*) – Drug names.

Returns **smiles** – Drug molecule structures represented by SMILES.

Return type list[str] or list[list[str,None]]

class pytrial.model_utils.drug.**DrugGraph**(*input_dir='./resources'*)

Bases: object

Provide tools to get hierarchy of drug by ACT codes.

From ATC2 - ATC4 (dont include ATC5)

preprocess(*dir='./resources'*)

Process raw data and deposit the graph data to the local disk. Search *ndc_atc.csv* under the given directory *dir*. If not, download the raw files to the disk and unzip.

CHAPTER

ELEVEN

UTILS

11.1 utils.trainer

class pytrial.utils.trainer.**Trainer**(*model: torch.nn.modules.module.Module*, *train_objectives: List[Tuple[torch.utils.data.dataloader.DataLoader, torch.nn.modules.module.Module]]*, *test_data=None*, *test_metric=None*, *less_is_better=False*, *load_best_at_end=True*, *n_gpus=1*, *output_dir='./checkpoints/'*, ***kwargs*)

Bases: object

A general trainer used to train deep learning models.

Parameters

- **model** (*nn.Module*) – The model to be trained.
- **train_objectives** (*list[tuple[DataLoader, nn.Module]]*) – The defined pairs of dataloaders and the loss models.
- **test_data** (*(optional) dict or Dataset*) – Depending on the implemented *get_test_dataloader* function. That function receives it as inputs and return test dataloader.
- **test_metric** (*(optional) str*) – Which test metric is used to judge the best checkpoint during the training. Only used when *test_data* is given. Should be contained in the returned metric dict by *evaluate* function.
- **less_is_better** (*(optional) bool*) – If the test metric is less the better. Ignored if no test_data and test_metric is given.
- **load_best_at_end** (*bool*) – If load the best checkpoint at the end of training.
- **n_gpus** (*int*) – How many GPUs used to kick of training. If set larger than 1, parallel training will be used.
- **output_dir** (*str*) – The intermediate model checkpoints during the training will be dump to under this dir.

Examples

```
>>> trainer = Trainer(
... model=model,
... train_objectives=[(dataloader1, loss_model1), (dataloader2, loss_model2)],
... )
>>> trainer.train(
... epochs=10,
... )
```

evaluate()
Need to be created by specific tasks.

Returns

Return type A dict of computed evalution metrics.

evaluated = False

get_test_dataloader(*test_data*)
Need to be created by specific tasks.

prepare_input(*data*)
Need to be reimplemented sometimes when input data is not in the standard dict structure.

train(*epochs=10*, *learning_rate=2e-05*, *weight_decay=0.0001*, *warmup_ratio=0*, *scheduler='warmupcosine'*, *evaluation_steps=10*, *max_grad_norm=0.5*, *use_amp=False*, ***kwargs*)
Kick of training using the provided loss model and train dataloaders.

Parameters

- **epochs** (*int (default=10)*) – Number of iterations (epochs) over the corpus.
- **learning_rate** (*float (default=3e-5)*) – The learning rate.
- **weight_decay** (*float (default=1e-4)*) – Weight decay applied for regularization.
- **warmup_ratio** (*float (default=0)*) – How many steps used for warmup training.

  If set 0, not warmup.
- **scheduler** (*{'constantlr','warmupconstant','warmuplinear','warmupcosine', 'warmupcosinewithhardrestarts'}*) – Pick learning rate scheduler for warmup. Ignored if warmup_ratio <= 0.
- **evaluation_steps** (*int (default=10)*) – How many iterations

  while we print the training loss and conduct evaluation if evaluator is given.
- **max_grad_norm** (*float (default=0.5)*) – Clip the gradient to avoid NaN.
- **use_amp** (*bool (default=False)*) – Whether or not use mixed precision training.

train_one_iteration(*max_grad_norm=None*, *warmup_steps=None*, *use_amp=None*, *scaler=None*, *train_loss_dict=None*)
Default training one iteration steps, can be subclass can reimplemented.

CHAPTER

TWELVE

ABOUT US

PyTrial team is affiliated with SunLab@UIUC. Our team members are:

- Zifeng Wang (PhD student @ UIUC): zifengw2 AT illinois DOT edu
- Brandon Theodorou (PhD student @ UIUC): bpt3 AT illinois DOT edu
- Tianfan Fu (PhD student @ Gatech): tfu42 AT gatech DOT edu
- Jingtang Ma (MS student @ UIUC): jm58 AT illinois DOT edu
- Jimeng Sun (Professor @ UIUC): jimeng AT illinois DOT edu

INDEX

M

N

P

R

S

T

U

V

W

X