APRILE

Hao Xu

Jan 17, 2022

GETTING STARTED:

1 Introduction 1

2 Installation 3

3 aprile.model 5

4 aprile.utils module 11

5 aprile.pure_query module 13

6 Indices and tables 15

Python Module Index 17

Index 19

CHAPTER

ONE

INTRODUCTION

Adverse Polypharmacy Reaction Intelligent Learner and Explainer (APRILE) is an explainable framework to reveal the mechanisms underlying adverse drug reactions (ADRs) caused by polypharmacy therapy. After learning from massive biomedical data, APRILE generate a small pharmacogenomic knowledge graph (i.e. drug targets and protein interactions) as mechanistic explanation for a drug-drug interaction (DDI) which associated an ADR and a set of such interactions.

1.1 Features

APRILE has three key features:

- Predicts side effects for drug combinations and gives the prediction reasons
- Delineate non-intuitive mechanistic associations between {genes, proteins, biological processes} and {symptoms, diseases, mental disorders ADRs}
- Explore molecular mechanisms for 843,318 (learned) + 93,966 (novel) side effect–drug pair events, spanning 861 side effects (472 diseases, 485 symptoms and 9 mental disorders) and 20 disease categories, have been suggested.

APRILE is able to answer the following example questions:

- Why the combination use of a pair of drugs (nicotine, ondansetron) causes anxiety?
- When taking fexofenadine, hydroxyzineand and loratadine simultaneously, what side effects may occur, and why?
- Which genes are associated with the infection diseases?
- What are the common mechanisms among peptic ulcers (such as duodenal ulcer, gastric ulcer and esophageal ulcer)?

We have demonstrated the viability of discovering polypharmacy side effect mechanisms by learning from an AI model trained on massive biomedical data (see our [paper])

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

PyKale requires a Python version 3.7 or above. Before installing `aprile`, PyTorch and PyTorch Geometric are required to be installed matching your hardware.

Note: We recommend using torch 1.4.0 (python3.7+cuda10.1), torch-cluster 1.5.4, torch-scatter 2.0.4, torch-sparse 0.6.1, torch-spline-cov 1.2.0 and torch-geometric 1.4.2

2.2 Pip install

Install the environment dependencies of APRILE using `pip` for the stable version:

```
pip install aprile
```

CHAPTER

THREE

APRILE.MODEL

class aprile.model.**Aprile**(*gdata*, *device='cpu'*)

Bases: object

APRILE: explaning polypharmacy side effect using a pre-trained APRILE-Pred model

Parameters **device** (*str*) – 'cpu' or 'cuda', for running APRILE-Explainer

enrich_go(*pp_left_index*)

gene ontology enrichment analysis

explain_list(*drug_list_1*, *drug_list_2*, *side_effect_list*, *regularization=2*, *if_auto_tuning=True*, *if_pred=True*)

generate explanation for a list of adverse drug events

explain_query(*query*, *if_auto_tuning=True*, *regularization=2*)

generate explanation for a AprileEQuery query

get_prediction_test(*threshold=0.5*)

generate predictions for DDIs in the testing set

Parameters **threshold** (*float, optional*) – the threshold of probability scores for DDIs. Defaults to 0.5.

Returns prediction results

Return type *AprileQuery*

get_prediction_train(*threshold=0.5*)

generate predictions for DDIs in the training set

Parameters **threshold** (*float, optional*) – the threshold of probability scores for DDIs. Defaults to 0.5.

Returns prediction results

Return type *AprileQuery*

predict(*drug1*, *drug2*, *side_effect*, *threshold=0.5*)

Predict the probability of DDIs

Parameters

- **drug1** (*list*) – a list of drug
- **drug2** (*list*) – a list of drug pairing *drug1*
- **side_effect** (*list*) – a list of side effect
- **threshold** (*float, optional*) – for probability. Defaults to 0.5.

Raises **ValueError** – None of DDIs meets the threshold

Returns prediction results

Return type *AprileQuery*

class aprile.model.**AprileExplainer**(*model*, *data*, *device*)
Bases: object

Explain APRILE-Predictor's predictions by given a small set of drug targets and protein-protein interactions

explain(*drug_list_1*, *drug_list_2*, *side_effect_list*, *regularization=1*)

class aprile.model.**AprileGCN**(*in_channels*, *out_channels*, *improved=False*, *cached=False*, *bias=True*, ***kwargs*)
Bases: torch_geometric.nn.conv.message_passing.MessagePassing

Graph convolutional neural network [1] with edge weights/masks. [1]: *"Semi-supervised Classification with Graph Convolutional Networks" <https://arxiv.org/abs/1609.02907>* (ICLR 2017) paper.

Note: For more information please see Pytorch Geomertic's nn.GCNConv docs.

Parameters

- **in_channels** (*int*) – size of each inputs samples
- **out_channels** (*[type]*) – size of each outputs samples
- **improved** (*bool, optional*) – Defaults to False.
- **cached** (*bool, optional*) – Defaults to False.
- **bias** (*bool, optional*) – Defaults to True.

forward(*x*, *edge_index*, *edge_weight=None*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

message(*x_j*, *norm*)
Constructs messages from node j to node i in analogy to ϕ_{Θ} for each edge in edge_index. This function can take any argument as input which was initially passed to propagate(). Furthermore, tensors passed to propagate() can be mapped to the respective nodes i and j by appending _i or _j to the variable name, *.e.g.* x_i and x_j.

static norm(*edge_index*, *num_nodes*, *edge_weight*, *improved=False*, *dtype=None*)
Add self-loops and apply symmetric normalization

reset_parameters()

update(*aggr_out*)
Updates node embeddings in analogy to γ_{Θ} for each node $i \in \mathcal{V}$. Takes in the output of aggregation as first argument and any argument which was initially passed to propagate().

class aprile.model.**AprilePredModel**(*pp*, *pd*, *mip*)
Bases: torch.nn.modules.module.Module

Aprile-Pred model structure

class aprile.model.**AprilePredictorPretrained**(*data_path*)
Bases: object

Make adverse polypharmacy reactions using a pre-trained predictor

Parameters **data_path** (*str*) – the path of pre-trained Aprile-Predictor

predict(*drug1*, *drug2*, *side_effect*, *device='cpu'*, *threshold=0.5*)

class aprile.model.**AprileQuery**(*drug1*, *drug2*, *side_effect*, *regularization=2*)
Bases: object

A class for quering APRILE's prediction and expalanition results

Parameters

- **drug1** (*list*) – a list of drug
- **drug2** (*list*) – a list of drug pairing with *drug1*
- **side_effect** (*list*) – a list of side effect caused by drug pairs
- **regularization** (*int, optional*) – the coefficient for control the size of explanation. Defaults to 2.

get_GOEnrich_table()
get Gene Ontology analysis results

Returns significant GOs, genes and additional mappings

Return type pandas.DataFrame

get_pred_table()
generate the prediction results

Returns DDIs, probability, PIU, PPIU and additional mappings

Return type pandas.DataFrame

get_query()
get query details

get_subgraph(*if_show=True*, *save_path=None*, *prot_graph_dict=None*, *drug_name_dict=None*)
Visualize explanation

Parameters

- **if_show** (*bool, optional*) – if print the figure. Defaults to True.
- **save_path** (*str, optional*) – the path to save figure. Defaults to None.

Returns DDIs and their mechanisms

Return type matplotlib.pyplot.figure()

static load_from_pkl(*file*)
load a query from a pickle file

Parameters **file** (*str*) – the file's path

Return type *AprileQuery*

set_enrich_result(*goea_results_sig*, *geneid2symbol*)

set_exp_result(*pp_index*, *pp_weight*, *pd_index*, *pd_weight*)

set_pred_result(*probability*, *piu_score*, *ppiu_score*, *drug_idx_to_id*, *drug_idx_to_name*, *side_effect_idx_to_name*)

to_pickle(*file*)
save the current query to a pickle file

Parameters **file** (*str*) – the path to save the object

class aprile.model.MultiInnerProductDecoder(*in_dim*, *num_et*)
Bases: torch.nn.modules.module.Module

DistMult tensor factorization for side effect prediction,

Parameters

- **in_dim** (*int*) – the dimension of drug feature
- **num_et** (*int*) – the number of side effect

forward(*z*, *edge_index*, *edge_type*, *sigmoid=True*)
forward propagation to predict {(drug, drug, side_effect)}

Parameters

- **z** (*torch.Tensor*) – drug features
- **edge_index** (*torch.Tensor*) – sparse representation of DDIs
- **edge_type** (*torch Tensor*) – side effect associated with each edge index
- **sigmoid** (*bool, optional*) – if apply Sigmoid. Defaults to True.

Returns probability of DDIs with associated

Return type torch tensor

class aprile.model.PD(*protein_dim*, *d_dim_prot*, *n_drug*, *d_dim_feat=32*)
Bases: torch.nn.modules.module.Module

Drug representation module

Parameters

- **protein_dim** (*int*) – the size of protein embeddings
- **d_dim_prot** (*int*) – the size of drug embeddings for the related pharmacogenomic information
- **n_drug** (*int*) – the number of drugs
- **d_dim_feat** (*int, optional*) – the size of drug feature embeddings. Defaults to 32.

forward(*x*, *pd_edge_index*, *edge_weight=None*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

reset_parameters()

class aprile.model.PP(*in_dim*, *nhid_list*)
Bases: torch.nn.modules.module.Module

Protein representation module

Parameters

- **in_dim** (*int*) – the size of each input samples
- **nhid_list** (*list*) – the size of each intermediary embeddings and outputs

forward(*x*, *pp_edge_index*, *edge_weight*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

reset_parameters()

class aprile.model.**Pre_mask**(*pp_n_link*, *pd_n_link*)

Bases: `torch.nn.modules.module.Module`

AprileGCN edge masker for AprileExplainer

Parameters

- **pp_n_link** (*int*) – the number of protein-protein edges
- **pd_n_link** (*int*) – the number of protein-drug edges

desaturate()

reset_parameters()

saturate()

CHAPTER

FOUR

APRILE.UTILS MODULE

aprile.utils.**args_parse_pred**(*drug_index_1*, *drug_index_2*, *side_effect_index*, *n_drug*, *n_side_effect*)

Parameters

- **drug_index_1** – char '*' or string of the format list of int, like 2,3,4
- **drug_index_2** – char '*' or string of the format list of int
- **side_effect_index** – char '*' or string of the format list of int

Returns three lists of int

aprile.utils.**args_parse_train**(*drug_index_1*, *drug_index_2*, *side_effect_index*, *rg*, *et*, *idx*)

Parameters

- **drug_index_1** – char '*' or string of the format list of int, like 2,3,4
- **drug_index_2** – char '*' or string of the format list of int
- **side_effect_index** – char '*' or string of the format list of int
- **rg** – int tensor of shape (n_side_effect, 2)
- **et** – int tensor of shape (n_dd_edge)
- **idx** – int tensor of shape (2, n_dd_edge)

Returns three lists of int

aprile.utils.**auprc_auroc_ap**(*target_tensor*, *score_tensor*)

aprile.utils.**dense_id**(*n*)

aprile.utils.**dict_ep_to_nparray**(*out_dict*, *epoch*)

aprile.utils.**get_edge_index_from_coo**(*mat*, *bidirection*)

aprile.utils.**get_indices_mask**(*indices*, *in_indices*)

aprile.utils.**get_range_list**(*edge_list*)

aprile.utils.**negative_sampling**(*pos_edge_index*, *num_nodes*)

aprile.utils.**normalize**(*input*)

aprile.utils.**process_edges**(*raw_edge_list*, *p=0.9*)

aprile.utils.**remove_bidirection**(*edge_index*, *edge_type*)

aprile.utils.**sparse_id**(*n*)

aprile.utils.**to_bidirection**(*edge_index*, *edge_type=None*)

aprile.utils.**typed_negative_sampling**(*pos_edge_index*, *num_nodes*, *range_list*)

aprile.utils.**uniform**(*size*, *tensor*)

aprile.utils.**visualize_graph**(*pp_idx*, *pp_weight*, *pd_idx*, *pd_weight*, *d1*, *d2*, *out_path*, *protein_name_dict=None*, *drug_name_dict=None*, *hiden=True*, *size=(40, 40)*)

visualize Aprile-Exp's outputs

1. use different color for pp and pd edges
2. annotate the weight of each edge near the edge (or annotate with the tranparentness of edges for each edge)
3. annotate the name of each node near the node, if name_dict=None, then annotate with node's index

Parameters

- **pp_idx** (*torch.Tensor*) – integer tensor (2, n_pp_edges)
- **pp_weight** (*torch.Tensor*) – float tensor (1, n_pp_edges), values with (0, 1)
- **pd_idx** (*torch.Tensor*) – integer tensor (2, n_pd_edges)
- **pd_weight** (*torch.Tensor*) – float tensor (1, n_pd_edges), values with (0, 1)
- **d1** (*list*) – drug list
- **d2** (*list*) – drug list pairing with *d1*
- **out_path** (*str*) – output path
- **protein_name_dict** (*dict, optional*) – the mapping for protein makers' text. Defaults to None.
- **drug_name_dict** (*dict, optional*) – the mapping for drug markers' text. Defaults to None.
- **hiden** (*bool, optional*) – if show related edge with edge weight of 0.01. Defaults to True.
- **size** (*tuple, optional*) – the figure size. Defaults to (40, 40).

Returns the graph object matplotlib.pyplot.figure: the ploted figure

Return type networkx.Graph

CHAPTER

FIVE

APRILE.PURE_QUERY MODULE

class aprile.pure_query.**PureAprileQuery**(*drug1*, *drug2*, *side_effect*, *gdata*, *regularization=2*)

Bases: object

get_GOEnrich_table()

get_pred_table()

get_query()

get_subgraph(*if_show=True*, *save_path=None*)

static load_from_pkl(*file*)

set_enrich_result(*goea_results_sig*)

set_exp_result(*pp_index*, *pp_weight*, *pd_index*, *pd_weight*)

set_pred_result(*probability*, *piu_score*, *ppiu_score*)

to_pickle(*file*)

aprile.pure_query.**visualize_graph**(*pp_idx*, *pp_weight*, *pd_idx*, *pd_weight*, *pp_adj*, *d1*, *d2*, *out_path*, *protein_name_dict=None*, *drug_name_dict=None*, *hiden=True*, *size=(40, 40)*)

Parameters

- **pp_idx** – integer tensor of the shape (2, n_pp_edges)
- **pp_weight** – float tensor of the shape (1, n_pp_edges), values within (0,1)
- **pd_idx** – integer tensor of the shape (2, n_pd_edges)
- **pd_weight** – float tensor of the shape (1, n_pd_edges), values within (0,1)
- **protein_name_dict** – store elements {protein_index -> protein name}
- **drug_name_dict** – store elements {drug_index -> drug name}

1. use different color for pp and pd edges
2. annotate the weight of each edge near the edge (or annotate with the tranparentness of edges for each edge)
3. annotate the name of each node near the node, if name_dict=None, then annotate with node's index

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

aprile.model, 5
aprile.pure_query, 13
aprile.utils, 11

INDEX

A

Aprile (*class in aprile.model*), 5
aprile.model
    module, 5
aprile.pure_query
    module, 13
aprile.utils
    module, 11
AprileExplainer (*class in aprile.model*), 6
AprileGCN (*class in aprile.model*), 6
AprilePredictorPretrained (*class in aprile.model*), 6
AprilePredModel (*class in aprile.model*), 6
AprileQuery (*class in aprile.model*), 7
args_parse_pred() (*in module aprile.utils*), 11
args_parse_train() (*in module aprile.utils*), 11
auprc_auroc_ap() (*in module aprile.utils*), 11

D

dense_id() (*in module aprile.utils*), 11
desaturate() (*aprile.model.Pre_mask method*), 9
dict_ep_to_nparray() (*in module aprile.utils*), 11

E

enrich_go() (*aprile.model.Aprile method*), 5
explain() (*aprile.model.AprileExplainer method*), 6
explain_list() (*aprile.model.Aprile method*), 5
explain_query() (*aprile.model.Aprile method*), 5

F

forward() (*aprile.model.AprileGCN method*), 6
forward() (*aprile.model.MultiInnerProductDecoder method*), 8
forward() (*aprile.model.PD method*), 8
forward() (*aprile.model.PP method*), 9

G

get_edge_index_from_coo() (*in module aprile.utils*), 11
get_GOEnrich_table() (*aprile.model.AprileQuery method*), 7
get_GOEnrich_table() (*aprile.pure_query.PureAprileQuery method*), 13
get_indices_mask() (*in module aprile.utils*), 11
get_pred_table() (*aprile.model.AprileQuery method*), 7
get_pred_table() (*aprile.pure_query.PureAprileQuery method*), 13
get_prediction_test() (*aprile.model.Aprile method*), 5
get_prediction_train() (*aprile.model.Aprile method*), 5
get_query() (*aprile.model.AprileQuery method*), 7
get_query() (*aprile.pure_query.PureAprileQuery method*), 13
get_range_list() (*in module aprile.utils*), 11
get_subgraph() (*aprile.model.AprileQuery method*), 7
get_subgraph() (*aprile.pure_query.PureAprileQuery method*), 13

L

load_from_pkl() (*aprile.model.AprileQuery static method*), 7
load_from_pkl() (*aprile.pure_query.PureAprileQuery static method*), 13

M

message() (*aprile.model.AprileGCN method*), 6
module
    aprile.model, 5
    aprile.pure_query, 13
    aprile.utils, 11
MultiInnerProductDecoder (*class in aprile.model*), 8

N

negative_sampling() (*in module aprile.utils*), 11
norm() (*aprile.model.AprileGCN static method*), 6
normalize() (*in module aprile.utils*), 11

P

PD (*class in aprile.model*), 8
PP (*class in aprile.model*), 8

Pre_mask (*class in aprile.model*), 9
predict() (*aprile.model.Aprile method*), 5
predict() (*aprile.model.AprilePredictorPretrained method*), 7
process_edges() (*in module aprile.utils*), 11
PureAprileQuery (*class in aprile.pure_query*), 13

R

remove_bidirection() (*in module aprile.utils*), 11
reset_parameters() (*aprile.model.AprileGCN method*), 6
reset_parameters() (*aprile.model.PD method*), 8
reset_parameters() (*aprile.model.PP method*), 9
reset_parameters() (*aprile.model.Pre_mask method*), 9

S

saturate() (*aprile.model.Pre_mask method*), 9
set_enrich_result() (*aprile.model.AprileQuery method*), 7
set_enrich_result() (*aprile.pure_query.PureAprileQuery method*), 13
set_exp_result() (*aprile.model.AprileQuery method*), 7
set_exp_result() (*aprile.pure_query.PureAprileQuery method*), 13
set_pred_result() (*aprile.model.AprileQuery method*), 7
set_pred_result() (*aprile.pure_query.PureAprileQuery method*), 13
sparse_id() (*in module aprile.utils*), 11

T

to_bidirection() (*in module aprile.utils*), 11
to_pickle() (*aprile.model.AprileQuery method*), 7
to_pickle() (*aprile.pure_query.PureAprileQuery method*), 13
typed_negative_sampling() (*in module aprile.utils*), 12

U

uniform() (*in module aprile.utils*), 12
update() (*aprile.model.AprileGCN method*), 6

V

visualize_graph() (*in module aprile.pure_query*), 13
visualize_graph() (*in module aprile.utils*), 12