

453 lines - 1 Removal

```

1 import os
2 import glob
3 import json
4 import torch
5 import pickle
6 import shutil
7 import numpy as np
8 import os.path as osp
9
10 from torch_geometric.datasets import MoleculeNet
11 from torch_geometric.utils import dense_to_sparse
12 from torch.utils.data import random_split, Subset
13 from torch_geometric.data import Data, InMemoryDataset,
14     download_url, extract_zip
15 from torch_geometric.loader import DataLoader
16
17 def undirected_graph(data):
18     data.edge_index = torch.cat([torch.stack([data.edge_
19         index[1], data.edge_index[0]], dim=0),
20         data.edge_index], dim=
21         1)
22     return data
23
24 def split(data, batch):
25     # i-th contains elements from slice[i] to slice[i+1]
26     node_slice = torch.cumsum(torch.from_numpy(np.bincou
27         nt(batch)), 0)
28     node_slice = torch.cat([torch.tensor([0]), node_slic
29         e])
30     row, _ = data.edge_index
31     edge_slice = torch.cumsum(torch.from_numpy(np.bincou
32         nt(batch[row])), 0)
33     edge_slice = torch.cat([torch.tensor([0]), edge_slic
34         e])
35
36     # Edge indices should start at zero for every graph.
37     data.edge_index -= node_slice[batch[row]].unsqueeze
38         (0)
39     data.__num_nodes__ = np.bincount(batch).tolist()
40
41     slices = dict()
42     slices['x'] = node_slice
43     slices['edge_index'] = edge_slice
44     slices['y'] = torch.arange(0, batch[-1] + 2, dtype=t
45         orch.long)
46     return data, slices
47
48 def read_file(folder, prefix, name):
49     file_path = osp.join(folder, prefix + f'_{name}.tx
50         t')
51     return np.genfromtxt(file_path, dtype=np.int64)
52
53 def read_sentiagraph_data(folder: str, prefix: str):
54     txt_files = glob.glob(os.path.join(folder, "{}_*.tx
55         t".format(prefix)))
56     json_files = glob.glob(os.path.join(folder, "{}_*.js
57         on".format(prefix)))
58     txt_names = [f.split(os.sep)[-1][len(prefix) + 1:-4]
59         for f in txt_files]

```

583 lines + 131 Additions

```

1 import os
2 import glob
3 import json
4 import torch
5 import pickle
6 import shutil
7 import numpy as np
8 import os.path as osp
9 from typing import Optional, Callable
10 from torch_geometric.datasets import MoleculeNet
11 from torch_geometric.utils import dense_to_sparse
12 from torch.utils.data import random_split, Subset
13 from torch_geometric.data import Data, InMemoryDataset,
14     download_url, extract_zip, extract_tar
15 from torch_geometric.loader import DataLoader
16
17 def undirected_graph(data):
18     data.edge_index = torch.cat([torch.stack([data.edge_
19         index[1], data.edge_index[0]], dim=0),
20         data.edge_index], dim=
21         1)
22     return data
23
24 def split(data, batch):
25     # i-th contains elements from slice[i] to slice[i+1]
26     node_slice = torch.cumsum(torch.from_numpy(np.bincou
27         nt(batch)), 0)
28     node_slice = torch.cat([torch.tensor([0]), node_slic
29         e])
30     row, _ = data.edge_index
31     edge_slice = torch.cumsum(torch.from_numpy(np.bincou
32         nt(batch[row])), 0)
33     edge_slice = torch.cat([torch.tensor([0]), edge_slic
34         e])
35
36     # Edge indices should start at zero for every graph.
37     data.edge_index -= node_slice[batch[row]].unsqueeze
38         (0)
39     data.__num_nodes__ = np.bincount(batch).tolist()
40
41     slices = dict()
42     slices['x'] = node_slice
43     slices['edge_index'] = edge_slice
44     slices['y'] = torch.arange(0, batch[-1] + 2, dtype=t
45         orch.long)
46     return data, slices
47
48 def read_file(folder, prefix, name):
49     file_path = osp.join(folder, prefix + f'_{name}.tx
50         t')
51     return np.genfromtxt(file_path, dtype=np.int64)
52
53 def read_sentiagraph_data(folder: str, prefix: str):
54     txt_files = glob.glob(os.path.join(folder, "{}_*.tx
55         t".format(prefix)))
56     json_files = glob.glob(os.path.join(folder, "{}_*.js
57         on".format(prefix)))
58     txt_names = [f.split(os.sep)[-1][len(prefix) + 1:-4]
59         for f in txt_files]

```

```

50     json_names = [f.split(os.sep)[-1][len(prefix) + 1:-
51 5] for f in json_files]
51     names = txt_names + json_names
52
53     with open(os.path.join(folder, prefix+"_node_feature
54 s.pkl"), 'rb') as f:
55         x: np.array = pickle.load(f)
56         x: torch.FloatTensor = torch.from_numpy(x)
57         edge_index: np.array = read_file(folder, prefix, 'ed
58 ge_index')
59         edge_index: torch.tensor = torch.tensor(edge_index,
60 dtype=torch.long).T
61         batch: np.array = read_file(folder, prefix, 'node_in
62 dicator') - 1      # from zero
63         y: np.array = read_file(folder, prefix, 'graph_label
64 s')
65         y: torch.tensor = torch.tensor(y, dtype=torch.long)
66         supplement = dict()
67         if 'split_indices' in names:
68             split_indices: np.array = read_file(folder, pref
69 ix, 'split_indices')
70             split_indices = torch.tensor(split_indices, dtyp
71 e=torch.long)
72             supplement['split_indices'] = split_indices
73         if 'sentence_tokens' in names:
74             with open(os.path.join(folder, prefix + '_senten
75 ce_tokens.json')) as f:
76                 sentence_tokens: dict = json.load(f)
77                 supplement['sentence_tokens'] = sentence_tokens
78
79         data = Data(x=x, edge_index=edge_index, y=y)
80         data, slices = split(data, batch)
81
82         return data, slices, supplement
83
84 def read_syn_data(folder: str, prefix):
85     with open(os.path.join(folder, f"{prefix}.pkl"), 'r
86 b') as f:
87         adj, features, y_train, y_val, y_test, train_mas
88 k, val_mask, test_mask, edge_label_matrix = pickle.load
89 (f)
90
91         x = torch.from_numpy(features).float()
92         y = train_mask.reshape(-1, 1) * y_train + val_mask.r
93 eshape(-1, 1) * y_val + test_mask.reshape(-1, 1) * y_tes
94 t
95         y = torch.from_numpy(np.where(y)[1])
96         edge_index = dense_to_sparse(torch.from_numpy(adj))
97         [0]
98         data = Data(x=x, y=y, edge_index=edge_index)
99         data.train_mask = torch.from_numpy(train_mask)
100         data.val_mask = torch.from_numpy(val_mask)
101         data.test_mask = torch.from_numpy(test_mask)
102         return data
103
104 def read_ba2motif_data(folder: str, prefix):
105     with open(os.path.join(folder, f"{prefix}.pkl"), 'r
106 b') as f:
107         dense_edges, node_features, graph_labels = pickl
108 e.load(f)
109
110         data_list = []
111         for graph_idx in range(dense_edges.shape[0]):

```

```

51     json_names = [f.split(os.sep)[-1][len(prefix) + 1:-
52 5] for f in json_files]
53     names = txt_names + json_names
54
55     with open(os.path.join(folder, prefix+"_node_feature
56 s.pkl"), 'rb') as f:
57         x: np.array = pickle.load(f)
58         x: torch.FloatTensor = torch.from_numpy(x)
59         edge_index: np.array = read_file(folder, prefix, 'ed
60 ge_index')
61         edge_index: torch.tensor = torch.tensor(edge_index,
62 dtype=torch.long).T
63         batch: np.array = read_file(folder, prefix, 'node_in
64 dicator') - 1      # from zero
65         y: np.array = read_file(folder, prefix, 'graph_label
66 s')
67         y: torch.tensor = torch.tensor(y, dtype=torch.long)
68         supplement = dict()
69         if 'split_indices' in names:
70             split_indices: np.array = read_file(folder, pref
71 ix, 'split_indices')
72             split_indices = torch.tensor(split_indices, dtyp
73 e=torch.long)
74             supplement['split_indices'] = split_indices
75         if 'sentence_tokens' in names:
76             with open(os.path.join(folder, prefix + '_senten
77 ce_tokens.json')) as f:
78                 sentence_tokens: dict = json.load(f)
79                 supplement['sentence_tokens'] = sentence_tokens
80
81         data = Data(x=x, edge_index=edge_index, y=y)
82         data, slices = split(data, batch)
83
84         return data, slices, supplement
85
86 def read_syn_data(folder: str, prefix):
87     with open(os.path.join(folder, f"{prefix}.pkl"), 'r
88 b') as f:
89         adj, features, y_train, y_val, y_test, train_mas
90 k, val_mask, test_mask, edge_label_matrix = pickle.load
91 (f)
92
93         x = torch.from_numpy(features).float()
94         y = train_mask.reshape(-1, 1) * y_train + val_mask.r
95 eshape(-1, 1) * y_val + test_mask.reshape(-1, 1) * y_tes
96 t
97         y = torch.from_numpy(np.where(y)[1])
98         edge_index = dense_to_sparse(torch.from_numpy(adj))
99         [0]
100         data = Data(x=x, y=y, edge_index=edge_index)
101         data.train_mask = torch.from_numpy(train_mask)
102         data.val_mask = torch.from_numpy(val_mask)
103         data.test_mask = torch.from_numpy(test_mask)
104         return data
105
106 def read_ba2motif_data(folder: str, prefix):
107     with open(os.path.join(folder, f"{prefix}.pkl"), 'r
108 b') as f:
109         dense_edges, node_features, graph_labels = pickl
110 e.load(f)
111
112         data_list = []
113         for graph_idx in range(dense_edges.shape[0]):

```

```

99         data_list.append(Data(x=torch.from_numpy(node_fe
atures[graph_idx]).float(),
100                               edge_index=dense_to_sparse
(torch.from_numpy(dense_edges[graph_idx]))[0],
101                               y=torch.from_numpy(np.wher
e(graph_labels[graph_idx])[0])))
102     return data_list
103
104
105 def get_dataset(dataset_dir, dataset_name, task=None):
106     sync_dataset_dict = {
107         'BA_2Motifs'.lower(): 'BA_2Motifs',
108         'BA_Shapes'.lower(): 'BA_shapes',
109         'BA_Community'.lower(): 'BA_Community',
110         'Tree_Cycle'.lower(): 'Tree_Cycle',
111         'Tree_Grids'.lower(): 'Tree_Grids',
112         'BA_LRP'.lower(): 'ba_lrp'
113     }
114
115     sentigraph_names = ['Graph_SST2', 'Graph_Twitter',
'Graph_SST5']
116     sentigraph_names = [name.lower() for name in sentigr
aph_names]
117     molecule_net_dataset_names = [name.lower() for name
in MoleculeNet.names.keys()]
118     if dataset_name.lower() == 'Mutagenicity'.lower():
119         return load_MUTAG(dataset_dir, 'mutagenicity')
120     elif dataset_name.lower() in sync_dataset_dict.keys
():
121         sync_dataset_filename = sync_dataset_dict[datase
t_name.lower()]
122         return load_syn_data(dataset_dir, sync_dataset_f
ilename)
123     elif dataset_name.lower() in molecule_net_dataset_na
mes:
124         return load_MoleculeNet(dataset_dir, dataset_nam
e, task)
125     elif dataset_name.lower() in sentigraph_names:
126         return load_SeniGraph(dataset_dir, dataset_name)
127
128     else:
129         raise NotImplementedError

```

```

100         data_list.append(Data(x=torch.from_numpy(node_fe
atures[graph_idx]).float(),
101                               edge_index=dense_to_sparse
(torch.from_numpy(dense_edges[graph_idx]))[0],
102                               y=torch.from_numpy(np.wher
e(graph_labels[graph_idx])[0])))
103     return data_list
104
105
106 def get_dataset(dataset_dir, dataset_name, task=None):
107     sync_dataset_dict = {
108         'BA_2Motifs'.lower(): 'BA_2Motifs',
109         'BA_Shapes'.lower(): 'BA_shapes',
110         'BA_Community'.lower(): 'BA_Community',
111         'Tree_Cycle'.lower(): 'Tree_Cycle',
112         'Tree_Grids'.lower(): 'Tree_Grids',
113         'BA_LRP'.lower(): 'ba_lrp'
114     }
115
116     sentigraph_names = ['Graph_SST2', 'Graph_Twitter',
'Graph_SST5']
117     sentigraph_names = [name.lower() for name in sentigr
aph_names]
118     molecule_net_dataset_names = [name.lower() for name
in MoleculeNet.names.keys()]
119     if dataset_name.lower() == 'Mutagenicity'.lower():
120         return load_MUTAG(dataset_dir, 'mutagenicity')
121     elif dataset_name.lower() in sync_dataset_dict.keys
():
122         sync_dataset_filename = sync_dataset_dict[datase
t_name.lower()]
123         return load_syn_data(dataset_dir, sync_dataset_f
ilename)
124     elif dataset_name.lower() in molecule_net_dataset_na
mes:
125         return load_MoleculeNet(dataset_dir, dataset_nam
e, task)
126     elif dataset_name.lower() in sentigraph_names:
127         return load_SeniGraph(dataset_dir, dataset_name)
128     elif dataset_name.lower() == "MalNetTiny".lower():
129         return load_MalNetTiny(dataset_dir, 'malnettin
y')
130
131     else:
132         raise NotImplementedError
133
134 class MalNetTiny(InMemoryDataset):
135     r"""The MalNet Tiny dataset from the
136     "A Large-Scale Database for Graph Representation Le
arning"
137     <https://openreview.net/pdf?id=1xDTDk3XPW>`_ paper.
138     :class:`MalNetTiny` contains 5,000 malicious and ben
ign software function
139     call graphs across 5 different types. Each graph con
tains at most 5k nodes.
140
141     Args:
142         root (str): Root directory where the dataset sho
uld be saved.
143         split (str, optional): If :obj:`"train"`, loads
the training dataset.
144         If :obj:`"val"`, loads the validation datase
t.
145         If :obj:`"trainval"`, loads the training and
validation dataset.
146         If :obj:`"test"`, loads the test dataset.

```

```

147         If :obj:`None`, loads the entire dataset.
148         (default: :obj:`None`)
149         transform (callable, optional): A function/trans
form that takes in an
150         :obj:`torch_geometric.data.Data` object and
returns a transformed
151         version. The data object will be transformed
before every access.
152         (default: :obj:`None`)
153         pre_transform (callable, optional): A function/t
ransform that takes in
154         an :obj:`torch_geometric.data.Data` object a
nd returns a
155         transformed version. The data object will be
transformed before
156         being saved to disk. (default: :obj:`None`)
157         pre_filter (callable, optional): A function that
takes in an
158         :obj:`torch_geometric.data.Data` object and
returns a boolean
159         value, indicating whether the data object sh
ould be included in the
160         final dataset. (default: :obj:`None`)
161         """
162
163         data_url = ('http://malnet.cc.gatech.edu/'
164                     'graph-data/malnet-graphs-tiny.tar.gz')
165         split_url = 'http://malnet.cc.gatech.edu/split-info/
split_info_tiny.zip'
166         splits = ['train', 'val', 'test']
167
168         def __init__(
169             self,
170             root: str,
171             name: str,
172             split: Optional[str] = None,
173             transform: Optional[Callable] = None,
174             pre_transform: Optional[Callable] = None,
175             pre_filter: Optional[Callable] = None,
176         ):
177             self.root = root
178             self.name = name.lower()
179             if split not in {'train', 'val', 'trainval', 'te
st', None}:
180                 raise ValueError(f'Split "{split}" found, bu
t expected either '
181                                   f'"train", "val", "trainva
1", "test" or None')
182             super().__init__(root, transform, pre_transform,
pre_filter)
183             self.data, self.slices = torch.load(self.process
ed_paths[0])
184
185             if split is not None:
186                 split_slices = torch.load(self.processed_pat
hs[1])
187                 if split == 'train':
188                     self._indices = range(split_slices[0], s
plit_slices[1])
189                 elif split == 'val':
190                     self._indices = range(split_slices[1], s
plit_slices[2])
191                 elif split == 'trainval':
192                     self._indices = range(split_slices[0], s
plit_slices[2])
193                 elif split == 'test':

```

```

194         self._indices = range(split_slices[2], s
plit_slices[3])
195
196     @property
197     def raw_dir(self):
198         return os.path.join(self.root, self.name, 'raw')
199
200     @property
201     def raw_file_names(self):
202         return ['malnet-graphs-tiny', osp.join('split_in
fo_tiny', 'type')]
203
204     @property
205     def processed_file_names(self):
206         return ['data.pt', 'split_slices.pt']
207
208     @property
209     def processed_dir(self):
210         return os.path.join(self.root, self.name, 'proce
ssed')
211
212     def download(self):
213         path = download_url(self.data_url, self.raw_dir)
214         extract_tar(path, self.raw_dir)
215         os.unlink(path)
216
217         path = download_url(self.split_url, self.raw_di
r)
218         extract_zip(path, self.raw_dir)
219         os.unlink(path)
220
221     def process(self):
222         y_map = {}
223         data_list = []
224         split_slices = [0]
225
226         for split in ['train', 'val', 'test']:
227             with open(osp.join(self.raw_paths[1], f'{spl
it}.txt'), 'r') as f:
228                 filenames = f.read().split('\n')[:-1]
229                 split_slices.append(split_slices[-1] + l
en(filenames))
230
231             for filename in filenames:
232                 path = osp.join(self.raw_paths[0], f'{fi
lename}.edgelist')
233                 malware_type = filename.split('/')[0]
234                 y = y_map.setdefault(malware_type, len(y
_map))
235
236                 with open(path, 'r') as f:
237                     edges = f.read().split('\n')[5:-1]
238
239                     edge_index = [[int(s) for s in edge.spli
t()] for edge in edges]
240                     edge_index = torch.tensor(edge_index).t
().contiguous()
241                     num_nodes = int(edge_index.max()) + 1
242                     data = Data(edge_index=edge_index, y=y,
num_nodes=num_nodes)
243                     data_list.append(data)
244
245             if self.pre_filter is not None:
246                 data_list = [data for data in data_list if s
elf.pre_filter(data)]
247
248             if self.pre_transform is not None:

```

```

130
131 class MUTAGDataset(InMemoryDataset):
132     def __init__(self, root, name, transform=None, pre_t
ransform=None):
133         self.root = root
134         self.name = name.lower()
135         super(MUTAGDataset, self).__init__(root, transfo
rm, pre_transform)
136         self.data, self.slices = torch.load(self.process
ed_paths[0])
137
138     def __len__(self):
139         return len(self.slices['x']) - 1
140
141     @property
142     def raw_dir(self):
143         return os.path.join(self.root, self.name, 'raw')
144
145     @property
146     def raw_file_names(self):
147         return ['Mutagenicity_A', 'Mutagenicity_graph_la
bels', 'Mutagenicity_graph_indicator', 'Mutagenicity_nod
e_labels']
148
149     @property
150     def processed_dir(self):
151         return os.path.join(self.root, self.name, 'proce
ssed')
152
153     @property
154     def processed_file_names(self):
155         return ['data.pt']
156
157     def download(self):
158         url = 'https://www.chrsmrrs.com/graphkerneldatas
ets'
159         folder = osp.join(self.root, self.name)
160         path = download_url(f'{url}/{self.name}.zip', fo
lder)
161         extract_zip(path, folder)
162         os.unlink(path)
163         shutil.rmtree(self.raw_dir)
164         os.rename(osp.join(folder, self.name), self.raw_
dir)
165
166     def process(self):
167         r"""Processes the dataset to the :obj:`self.proc
essed_dir` folder."""
168         with open(os.path.join(self.raw_dir, 'Mutagenici
ty_node_labels.txt'), 'r') as f:
169             nodes_all_temp = f.read().splitlines()
170             nodes_all = [int(i) for i in nodes_all_temp]
171
172             adj_all = np.zeros((len(nodes_all), len(nodes_al
l)))
173             with open(os.path.join(self.raw_dir, 'Mutagenici
ty_A.txt'), 'r') as f:
174                 adj_list = f.read().splitlines()
175                 for item in adj_list:

```

```

249         data_list = [self.pre_transform(data) for da
ta in data_list]
250
251         torch.save(self.collate(data_list), self.process
ed_paths[0])
252         torch.save(split_slices, self.processed_paths
[1])
253
254
255 class MUTAGDataset(InMemoryDataset):
256     def __init__(self, root, name, transform=None, pre_t
ransform=None):
257         self.root = root
258         self.name = name.lower()
259         super(MUTAGDataset, self).__init__(root, transfo
rm, pre_transform)
260         self.data, self.slices = torch.load(self.process
ed_paths[0])
261
262     def __len__(self):
263         return len(self.slices['x']) - 1
264
265     @property
266     def raw_dir(self):
267         return os.path.join(self.root, self.name, 'raw')
268
269     @property
270     def raw_file_names(self):
271         return ['Mutagenicity_A', 'Mutagenicity_graph_la
bels', 'Mutagenicity_graph_indicator', 'Mutagenicity_nod
e_labels']
272
273     @property
274     def processed_dir(self):
275         return os.path.join(self.root, self.name, 'proce
ssed')
276
277     @property
278     def processed_file_names(self):
279         return ['data.pt']
280
281     def download(self):
282         url = 'https://www.chrsmrrs.com/graphkerneldatas
ets'
283         folder = osp.join(self.root, self.name)
284         path = download_url(f'{url}/{self.name}.zip', fo
lder)
285         extract_zip(path, folder)
286         os.unlink(path)
287         shutil.rmtree(self.raw_dir)
288         os.rename(osp.join(folder, self.name), self.raw_
dir)
289
290     def process(self):
291         r"""Processes the dataset to the :obj:`self.proc
essed_dir` folder."""
292         with open(os.path.join(self.raw_dir, 'Mutagenici
ty_node_labels.txt'), 'r') as f:
293             nodes_all_temp = f.read().splitlines()
294             nodes_all = [int(i) for i in nodes_all_temp]
295
296             adj_all = np.zeros((len(nodes_all), len(nodes_al
l)))
297             with open(os.path.join(self.raw_dir, 'Mutagenici
ty_A.txt'), 'r') as f:
298                 adj_list = f.read().splitlines()
299                 for item in adj_list:
300

```

```

176         lr = item.split(', ')
177         l = int(lr[0])
178         r = int(lr[1])
179         adj_all[l - 1, r - 1] = 1
180
181         with open(os.path.join(self.raw_dir, 'Mutagenici
ty_graph_indicator.txt'), 'r') as f:
182             graph_indicator_temp = f.read().splitlines()
183             graph_indicator = [int(i) for i in graph_ind
icator_temp]
184             graph_indicator = np.array(graph_indicator)
185
186         with open(os.path.join(self.raw_dir, 'Mutagenici
ty_graph_labels.txt'), 'r') as f:
187             graph_labels_temp = f.read().splitlines()
188             graph_labels = [int(i) for i in graph_labels
_temp]
189
190         data_list = []
191         for i in range(1, 4338):
192             idx = np.where(graph_indicator == i)
193             graph_len = len(idx[0])
194             adj = adj_all[idx[0][0]:idx[0][0] + graph_le
n, idx[0][0]:idx[0][0] + graph_len]
195             label = int(graph_labels[i - 1] == 1)
196             feature = nodes_all[idx[0][0]:idx[0][0] + gr
aph_len]
197             nb_cls = 14
198             targets = np.array(feature).reshape(-1)
199             one_hot_feature = np.eye(nb_cls)[targets]
200             data_example = Data(x=torch.from_numpy(one_h
ot_feature).float()),
201                                edge_index=dense_to_spar
se(torch.from_numpy(adj))[0],
202                                y=label)
203             data_list.append(data_example)
204
205         torch.save(self.collate(data_list), self.process
ed_paths[0])
206
207
208     class SentiGraphDataset(InMemoryDataset):
209         def __init__(self, root, name, transform=None, pre_t
ransform=undirected_graph):
210             self.name = name
211             super(SentiGraphDataset, self).__init__(root, tr
ansform, pre_transform)
212             self.data, self.slices, self.supplement = torch.
load(self.processed_paths[0])
213
214             @property
215             def raw_dir(self):
216                 return osp.join(self.root, self.name, 'raw')
217
218             @property
219             def processed_dir(self):
220                 return osp.join(self.root, self.name, 'processe
d')
221
222             @property
223             def raw_file_names(self):
224                 return ['node_features', 'node_indicator', 'sent
ence_tokens', 'edge_index',
225                         'graph_labels', 'split_indices']
226
227             @property
228             def processed_file_names(self):

```

```

301         lr = item.split(', ')
302         l = int(lr[0])
303         r = int(lr[1])
304         adj_all[l - 1, r - 1] = 1
305
306         with open(os.path.join(self.raw_dir, 'Mutagenici
ty_graph_indicator.txt'), 'r') as f:
307             graph_indicator_temp = f.read().splitlines()
308             graph_indicator = [int(i) for i in graph_ind
icator_temp]
309             graph_indicator = np.array(graph_indicator)
310
311         with open(os.path.join(self.raw_dir, 'Mutagenici
ty_graph_labels.txt'), 'r') as f:
312             graph_labels_temp = f.read().splitlines()
313             graph_labels = [int(i) for i in graph_labels
_temp]
314
315         data_list = []
316         for i in range(1, 4338):
317             idx = np.where(graph_indicator == i)
318             graph_len = len(idx[0])
319             adj = adj_all[idx[0][0]:idx[0][0] + graph_le
n, idx[0][0]:idx[0][0] + graph_len]
320             label = int(graph_labels[i - 1] == 1)
321             feature = nodes_all[idx[0][0]:idx[0][0] + gr
aph_len]
322             nb_cls = 14
323             targets = np.array(feature).reshape(-1)
324             one_hot_feature = np.eye(nb_cls)[targets]
325             data_example = Data(x=torch.from_numpy(one_h
ot_feature).float()),
326                                edge_index=dense_to_spar
se(torch.from_numpy(adj))[0],
327                                y=label)
328             data_list.append(data_example)
329
330         torch.save(self.collate(data_list), self.process
ed_paths[0])
331
332
333     class SentiGraphDataset(InMemoryDataset):
334         def __init__(self, root, name, transform=None, pre_t
ransform=undirected_graph):
335             self.name = name
336             super(SentiGraphDataset, self).__init__(root, tr
ansform, pre_transform)
337             self.data, self.slices, self.supplement = torch.
load(self.processed_paths[0])
338
339             @property
340             def raw_dir(self):
341                 return osp.join(self.root, self.name, 'raw')
342
343             @property
344             def processed_dir(self):
345                 return osp.join(self.root, self.name, 'processe
d')
346
347             @property
348             def raw_file_names(self):
349                 return ['node_features', 'node_indicator', 'sent
ence_tokens', 'edge_index',
350                         'graph_labels', 'split_indices']
351
352             @property
353             def processed_file_names(self):

```

```

229         return ['data.pt']
230
231     def process(self):
232         # Read data into huge `Data` list.
233         self.data, self.slices, self.supplement \
234             = read_sentigraph_data(self.raw_dir, self.
name)
235
236         if self.pre_filter is not None:
237             data_list = [self.get(idx) for idx in range
(len(self))]
238             data_list = [data for data in data_list if s
elf.pre_filter(data)]
239             self.data, self.slices = self.collate(data_l
ist)
240
241         if self.pre_transform is not None:
242             data_list = [self.get(idx) for idx in range
(len(self))]
243             data_list = [self.pre_transform(data) for da
ta in data_list]
244             self.data, self.slices = self.collate(data_l
ist)
245             torch.save((self.data, self.slices, self.supplem
ent), self.processed_paths[0])
246
247
248 class SynGraphDataset(InMemoryDataset):
249     def __init__(self, root, name, transform=None, pre_t
ransform=None):
250         self.name = name
251         super(SynGraphDataset, self).__init__(root, tran
sform, pre_transform)
252         self.data, self.slices = torch.load(self.process
ed_paths[0])
253
254     @property
255     def raw_dir(self):
256         return osp.join(self.root, self.name, 'raw')
257
258     @property
259     def processed_dir(self):
260         return osp.join(self.root, self.name, 'processe
d')
261
262     @property
263     def raw_file_names(self):
264         return [f"{self.name}.pkl"]
265
266     @property
267     def processed_file_names(self):
268         return ['data.pt']
269
270     def process(self):
271         # Read data into huge `Data` list.
272         data = read_syn_data(self.raw_dir, self.name)
273         data = data if self.pre_transform is None else s
elf.pre_transform(data)
274         torch.save(self.collate([data]), self.processed_
paths[0])
275
276
277 class BA2MotifDataset(InMemoryDataset):
278     def __init__(self, root, name, transform=None, pre_t
ransform=None):
279         self.name = name
280         super(BA2MotifDataset, self).__init__(root, tran
sform, pre_transform)

```

```

354         return ['data.pt']
355
356     def process(self):
357         # Read data into huge `Data` list.
358         self.data, self.slices, self.supplement \
359             = read_sentigraph_data(self.raw_dir, self.
name)
360
361         if self.pre_filter is not None:
362             data_list = [self.get(idx) for idx in range
(len(self))]
363             data_list = [data for data in data_list if s
elf.pre_filter(data)]
364             self.data, self.slices = self.collate(data_l
ist)
365
366         if self.pre_transform is not None:
367             data_list = [self.get(idx) for idx in range
(len(self))]
368             data_list = [self.pre_transform(data) for da
ta in data_list]
369             self.data, self.slices = self.collate(data_l
ist)
370             torch.save((self.data, self.slices, self.supplem
ent), self.processed_paths[0])
371
372
373 class SynGraphDataset(InMemoryDataset):
374     def __init__(self, root, name, transform=None, pre_t
ransform=None):
375         self.name = name
376         super(SynGraphDataset, self).__init__(root, tran
sform, pre_transform)
377         self.data, self.slices = torch.load(self.process
ed_paths[0])
378
379     @property
380     def raw_dir(self):
381         return osp.join(self.root, self.name, 'raw')
382
383     @property
384     def processed_dir(self):
385         return osp.join(self.root, self.name, 'processe
d')
386
387     @property
388     def raw_file_names(self):
389         return [f"{self.name}.pkl"]
390
391     @property
392     def processed_file_names(self):
393         return ['data.pt']
394
395     def process(self):
396         # Read data into huge `Data` list.
397         data = read_syn_data(self.raw_dir, self.name)
398         data = data if self.pre_transform is None else s
elf.pre_transform(data)
399         torch.save(self.collate([data]), self.processed_
paths[0])
400
401
402 class BA2MotifDataset(InMemoryDataset):
403     def __init__(self, root, name, transform=None, pre_t
ransform=None):
404         self.name = name
405         super(BA2MotifDataset, self).__init__(root, tran
sform, pre_transform)

```



```

281         self.data, self.slices = torch.load(self.process
ed_paths[0])
282
283     @property
284     def raw_dir(self):
285         return osp.join(self.root, self.name, 'raw')
286
287     @property
288     def processed_dir(self):
289         return osp.join(self.root, self.name, 'processe
d')
290
291     @property
292     def raw_file_names(self):
293         return [f"{self.name}.pkl"]
294
295     @property
296     def processed_file_names(self):
297         return ['data.pt']
298
299     def process(self):
300         # Read data into huge `Data` list.
301         data_list = read_ba2motif_data(self.raw_dir, sel
f.name)
302
303         if self.pre_filter is not None:
304             data_list = [self.get(idx) for idx in range
(len(self))]
305             data_list = [data for data in data_list if s
elf.pre_filter(data)]
306             self.data, self.slices = self.collate(data_l
ist)
307
308         if self.pre_transform is not None:
309             data_list = [self.get(idx) for idx in range
(len(self))]
310             data_list = [self.pre_transform(data) for da
ta in data_list]
311             self.data, self.slices = self.collate(data_l
ist)
312
313         torch.save(self.collate(data_list), self.process
ed_paths[0])
314
315
316 def load_MUTAG(dataset_dir, dataset_name):
317     """ 188 molecules where label = 1 denotes mutagenic
effect """
318     dataset = MUTAGDataset(root=dataset_dir, name=datase
t_name)
319     return dataset
320
321
322 class BA_LRP(InMemoryDataset):
323
324     def __init__(self, root, num_per_class, transform=No
ne, pre_transform=None):
325         self.num_per_class = num_per_class
326         super().__init__(root, transform, pre_transform)
327         self.data, self.slices = torch.load(self.process
ed_paths[0])
328
329     @property

```

```

406         self.data, self.slices = torch.load(self.process
ed_paths[0])
407
408     @property
409     def raw_dir(self):
410         return osp.join(self.root, self.name, 'raw')
411
412     @property
413     def processed_dir(self):
414         return osp.join(self.root, self.name, 'processe
d')
415
416     @property
417     def raw_file_names(self):
418         return [f"{self.name}.pkl"]
419
420     @property
421     def processed_file_names(self):
422         return ['data.pt']
423
424     def process(self):
425         # Read data into huge `Data` list.
426         data_list = read_ba2motif_data(self.raw_dir, sel
f.name)
427
428         if self.pre_filter is not None:
429             data_list = [self.get(idx) for idx in range
(len(self))]
430             data_list = [data for data in data_list if s
elf.pre_filter(data)]
431             self.data, self.slices = self.collate(data_l
ist)
432
433         if self.pre_transform is not None:
434             data_list = [self.get(idx) for idx in range
(len(self))]
435             data_list = [self.pre_transform(data) for da
ta in data_list]
436             self.data, self.slices = self.collate(data_l
ist)
437
438         torch.save(self.collate(data_list), self.process
ed_paths[0])
439
440
441 def load_MUTAG(dataset_dir, dataset_name):
442     """ 188 molecules where label = 1 denotes mutagenic
effect """
443     dataset = MUTAGDataset(root=dataset_dir, name=datase
t_name)
444     return dataset
445
446 def load_MalNetTiny(dataset_dir, dataset_name):
447
448     dataset = MalNetTiny(root=dataset_dir, name=dataset_
name)
449     return dataset
450
451
452 class BA_LRP(InMemoryDataset):
453
454     def __init__(self, root, num_per_class, transform=No
ne, pre_transform=None):
455         self.num_per_class = num_per_class
456         super().__init__(root, transform, pre_transform)
457         self.data, self.slices = torch.load(self.process
ed_paths[0])
458
459     @property

```

```

330     def processed_file_names(self):
331         return [f'data{self.num_per_class}.pt']
332
333     def gen_class1(self):
334         x = torch.tensor([[1], [1]], dtype=torch.float)
335         edge_index = torch.tensor([[0, 1], [1, 0]], dtype=torch.long)
336         data = Data(x=x, edge_index=edge_index, y=torch.tensor([[0]], dtype=torch.float))
337
338         for i in range(2, 20):
339             data.x = torch.cat([data.x, torch.tensor([[1]], dtype=torch.float)], dim=0)
340             deg = torch.stack([(data.edge_index[0] == node_idx).float().sum() for node_idx in range(i)], dim=0)
341             sum_deg = deg.sum(dim=0, keepdim=True)
342             probs = (deg / sum_deg).unsqueeze(0)
343             prob_dist = torch.distributions.Categorical(probs)
344             node_pick = prob_dist.sample().squeeze()
345             data.edge_index = torch.cat([data.edge_index,
346                                         torch.tensor([[node_pick, i], [i, node_pick]], dtype=torch.long)], dim=1)
347
348         return data
349
350     def gen_class2(self):
351         x = torch.tensor([[1], [1]], dtype=torch.float)
352         edge_index = torch.tensor([[0, 1], [1, 0]], dtype=torch.long)
353         data = Data(x=x, edge_index=edge_index, y=torch.tensor([[1]], dtype=torch.float))
354         epsilon = 1e-30
355
356         for i in range(2, 20):
357             data.x = torch.cat([data.x, torch.tensor([[1]], dtype=torch.float)], dim=0)
358             deg_reciprocal = torch.stack([1 / ((data.edge_index[0] == node_idx).float().sum() + epsilon) for node_idx in range(i)], dim=0)
359             sum_deg_reciprocal = deg_reciprocal.sum(dim=0, keepdim=True)
360             probs = (deg_reciprocal / sum_deg_reciprocal).unsqueeze(0)
361             prob_dist = torch.distributions.Categorical(probs)
362             node_pick = -1
363             for _ in range(1 if i % 5 != 4 else 2):
364                 new_node_pick = prob_dist.sample().squeeze()
365                 while new_node_pick == node_pick:
366                     new_node_pick = prob_dist.sample().squeeze()
367             node_pick = new_node_pick
368             data.edge_index = torch.cat([data.edge_index,
369                                         torch.tensor([[node_pick, i], [i, node_pick]], dtype=torch.long)], dim=1)
370
371         return data
372
373     def process(self):
374         data_list = []
375         for i in range(self.num_per_class):

```

```

460     def processed_file_names(self):
461         return [f'data{self.num_per_class}.pt']
462
463     def gen_class1(self):
464         x = torch.tensor([[1], [1]], dtype=torch.float)
465         edge_index = torch.tensor([[0, 1], [1, 0]], dtype=torch.long)
466         data = Data(x=x, edge_index=edge_index, y=torch.tensor([[0]], dtype=torch.float))
467
468         for i in range(2, 20):
469             data.x = torch.cat([data.x, torch.tensor([[1]], dtype=torch.float)], dim=0)
470             deg = torch.stack([(data.edge_index[0] == node_idx).float().sum() for node_idx in range(i)], dim=0)
471             sum_deg = deg.sum(dim=0, keepdim=True)
472             probs = (deg / sum_deg).unsqueeze(0)
473             prob_dist = torch.distributions.Categorical(probs)
474             node_pick = prob_dist.sample().squeeze()
475             data.edge_index = torch.cat([data.edge_index,
476                                         torch.tensor([[node_pick, i], [i, node_pick]], dtype=torch.long)], dim=1)
477
478         return data
479
480     def gen_class2(self):
481         x = torch.tensor([[1], [1]], dtype=torch.float)
482         edge_index = torch.tensor([[0, 1], [1, 0]], dtype=torch.long)
483         data = Data(x=x, edge_index=edge_index, y=torch.tensor([[1]], dtype=torch.float))
484         epsilon = 1e-30
485
486         for i in range(2, 20):
487             data.x = torch.cat([data.x, torch.tensor([[1]], dtype=torch.float)], dim=0)
488             deg_reciprocal = torch.stack([1 / ((data.edge_index[0] == node_idx).float().sum() + epsilon) for node_idx in range(i)], dim=0)
489             sum_deg_reciprocal = deg_reciprocal.sum(dim=0, keepdim=True)
490             probs = (deg_reciprocal / sum_deg_reciprocal).unsqueeze(0)
491             prob_dist = torch.distributions.Categorical(probs)
492             node_pick = -1
493             for _ in range(1 if i % 5 != 4 else 2):
494                 new_node_pick = prob_dist.sample().squeeze()
495                 while new_node_pick == node_pick:
496                     new_node_pick = prob_dist.sample().squeeze()
497             node_pick = new_node_pick
498             data.edge_index = torch.cat([data.edge_index,
499                                         torch.tensor([[node_pick, i], [i, node_pick]], dtype=torch.long)], dim=1)
500
501         return data
502
503     def process(self):
504         data_list = []
505         for i in range(self.num_per_class):

```

```

376         data_list.append(self.gen_class1())
377         data_list.append(self.gen_class2())
378
379         data, slices = self.collate(data_list)
380         torch.save((data, slices), self.processed_paths
[0])
381
382
383 def load_syn_data(dataset_dir, dataset_name):
384     """ The synthetic dataset """
385     if dataset_name.lower() == 'BA_2Motifs'.lower():
386         dataset = BA2MotifDataset(root=dataset_dir, name
=dataset_name)
387     elif dataset_name.lower() == 'BA_LRP'.lower():
388         dataset = BA_LRP(root=os.path.join(dataset_dir,
'ba_lrp'), num_per_class=10000)
389     else:
390         dataset = SynGraphDataset(root=dataset_dir, name
=dataset_name)
391     dataset.node_type_dict = {k: v for k, v in enumerate
(range(dataset.num_classes))}
392     dataset.node_color = None
393     return dataset
394
395
396 def load_MoleculeNet(dataset_dir, dataset_name, task=Non
e):
397     """ Attention the multi-task problems not solved yet
"""
398     molecule_net_dataset_names = {name.lower(): name for
name in MoleculeNet.names.keys()}
399     dataset = MoleculeNet(root=dataset_dir, name=molecul
e_net_dataset_names[dataset_name.lower()])
400     dataset.data.x = dataset.data.x.float()
401     if task is None:
402         dataset.data.y = dataset.data.y.squeeze().long()
403     else:
404         dataset.data.y = dataset.data.y[task].long()
405     dataset.node_type_dict = None
406     dataset.node_color = None
407     return dataset
408
409
410 def load_SeniGraph(dataset_dir, dataset_name):
411     dataset = SentiGraphDataset(root=dataset_dir, name=d
ataset_name)
412     return dataset
413
414
415 def get_dataloader(dataset, batch_size, random_split_flg
g=True, data_split_ratio=None, seed=2):
416     """
417     Args:
418         dataset:
419         batch_size: int
420         random_split_flag: bool
421         data_split_ratio: list, training, validation and
testing ratio
422         seed: random seed to split the dataset randomly
423     Returns:
424         a dictionary of training, validation, and testin
g DataLoader
425     """
426
427     if not random_split_flag and hasattr(dataset, 'suppl
ement'):

```

```

506         data_list.append(self.gen_class1())
507         data_list.append(self.gen_class2())
508
509         data, slices = self.collate(data_list)
510         torch.save((data, slices), self.processed_paths
[0])
511
512
513 def load_syn_data(dataset_dir, dataset_name):
514     """ The synthetic dataset """
515     if dataset_name.lower() == 'BA_2Motifs'.lower():
516         dataset = BA2MotifDataset(root=dataset_dir, name
=dataset_name)
517     elif dataset_name.lower() == 'BA_LRP'.lower():
518         dataset = BA_LRP(root=os.path.join(dataset_dir,
'ba_lrp'), num_per_class=10000)
519     else:
520         dataset = SynGraphDataset(root=dataset_dir, name
=dataset_name)
521     dataset.node_type_dict = {k: v for k, v in enumerate
(range(dataset.num_classes))}
522     dataset.node_color = None
523     return dataset
524
525
526 def load_MoleculeNet(dataset_dir, dataset_name, task=Non
e):
527     """ Attention the multi-task problems not solved yet
"""
528     molecule_net_dataset_names = {name.lower(): name for
name in MoleculeNet.names.keys()}
529     dataset = MoleculeNet(root=dataset_dir, name=molecul
e_net_dataset_names[dataset_name.lower()])
530     dataset.data.x = dataset.data.x.float()
531     if task is None:
532         dataset.data.y = dataset.data.y.squeeze().long()
533     else:
534         dataset.data.y = dataset.data.y[task].long()
535     dataset.node_type_dict = None
536     dataset.node_color = None
537     return dataset
538
539
540 def load_SeniGraph(dataset_dir, dataset_name):
541     dataset = SentiGraphDataset(root=dataset_dir, name=d
ataset_name)
542     return dataset
543
544
545 def get_dataloader(dataset, batch_size, random_split_flg
g=True, data_split_ratio=None, seed=2):
546     """
547     Args:
548         dataset:
549         batch_size: int
550         random_split_flag: bool
551         data_split_ratio: list, training, validation and
testing ratio
552         seed: random seed to split the dataset randomly
553     Returns:
554         a dictionary of training, validation, and testin
g DataLoader
555     """
556
557     if not random_split_flag and hasattr(dataset, 'suppl
ement'):

```

```

428         assert 'split_indices' in dataset.supplement.keys(), "split idx"
429         split_indices = dataset.supplement['split_indices']
430         train_indices = torch.where(split_indices == 0)
431         dev_indices = torch.where(split_indices == 1)
432         test_indices = torch.where(split_indices == 2)
433         [0].numpy().tolist()
434         train = Subset(dataset, train_indices)
435         eval = Subset(dataset, dev_indices)
436         test = Subset(dataset, test_indices)
437     else:
438         num_train = int(data_split_ratio[0] * len(dataset))
439         num_eval = int(data_split_ratio[1] * len(dataset))
440         num_test = len(dataset) - num_train - num_eval
441         train, eval, test = random_split(dataset, lengths=[num_train, num_eval, num_test],
442                                         generator=torch
443                                         h.Generator().manual_seed(seed))
444     dataloader = dict()
445     dataloader['train'] = DataLoader(train, batch_size=batch_size, shuffle=True)
446     dataloader['eval'] = DataLoader(eval, batch_size=batch_size, shuffle=False)
447     dataloader['test'] = DataLoader(test, batch_size=batch_size, shuffle=False)
448     return dataloader
449
450
451
452 if __name__ == '__main__':
453     get_dataset(dataset_dir='./datasets', dataset_name='bbbp')

```

```

558         assert 'split_indices' in dataset.supplement.keys(), "split idx"
559         split_indices = dataset.supplement['split_indices']
560         train_indices = torch.where(split_indices == 0)
561         dev_indices = torch.where(split_indices == 1)
562         test_indices = torch.where(split_indices == 2)
563         [0].numpy().tolist()
564         train = Subset(dataset, train_indices)
565         eval = Subset(dataset, dev_indices)
566         test = Subset(dataset, test_indices)
567     else:
568         num_train = int(data_split_ratio[0] * len(dataset))
569         num_eval = int(data_split_ratio[1] * len(dataset))
570         num_test = len(dataset) - num_train - num_eval
571         train, eval, test = random_split(dataset, lengths=[num_train, num_eval, num_test],
572                                         generator=torch
573                                         h.Generator().manual_seed(seed))
574     dataloader = dict()
575     dataloader['train'] = DataLoader(train, batch_size=batch_size, shuffle=True)
576     dataloader['eval'] = DataLoader(eval, batch_size=batch_size, shuffle=False)
577     dataloader['test'] = DataLoader(test, batch_size=batch_size, shuffle=False)
578     return dataloader
579
580
581
582 if __name__ == '__main__':
583     get_dataset(dataset_dir='./datasets', dataset_name='bbbp')

```