The GC19HG Project

Preprocessing and Model Development

Chris Winsor 5/2023

GC19HG (GeoCovid19 Heterogeneous Graph) is a social media research project that seeks to understand how the micro-level behaviors of millions of users combine into macro-level topic trends. The work uses the CrisisNLP GeoCov19 dataset - a set of tweets relating to Covid19 from February to May of 2020. The dataset can be found at (https://crisisnlp.qcri.org/covid19).

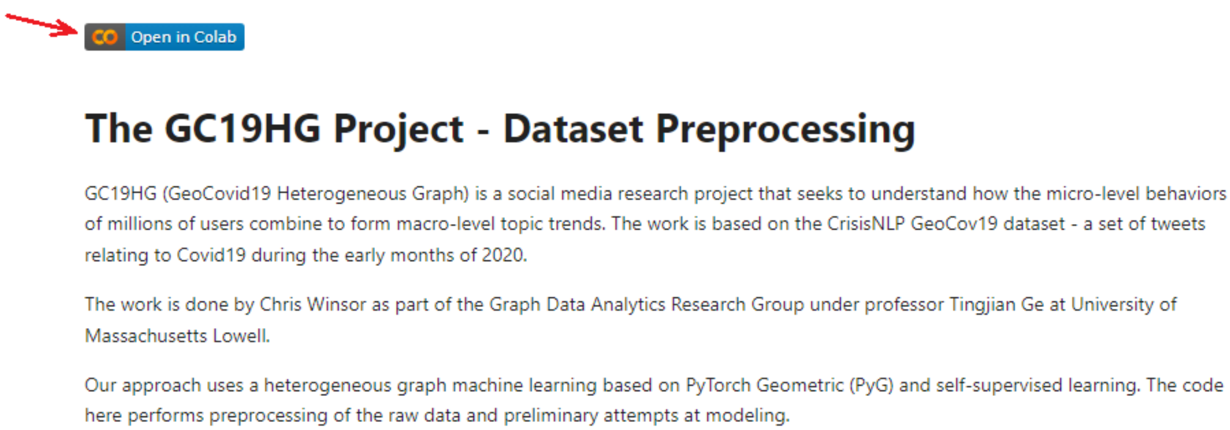
Our approach combines heterogeneous graph machine learning, NLP pre-trained models and self-supervised learning using the PyTorch Geometric (PyG) library. The code here performs preprocessing of the raw data and establishes an initial modeling attempt.

The code can be run from a Google Colab notebook with launcher at
https://github.com/cwinsor/uml_twitter/blob/main/colab_notebook_launcher.ipynb

Instructions to Run:

The best and easiest way is using Google Colab.

1. From a browser open the following link:
   https://github.com/cwinsor/uml_twitter/blob/main/colab_notebook_launcher.ipynb
2. Click the "Open in Colab" button.



3. Execute the blocks in sequence using the <ctrl>Enter keystroke combination.

For Developers

Versions and platform are in Appendix 1.

Two primary files are:

- g40_preprocess.py: parse and filter (a.k.a. "preprocessing")
- g41_train_test.py: model, dataset, training

Preprocessing

Parse

The parse step reads the raw .ijson Twitter files, identifies re-tweets, and extracts out the desired data. The desired data is retweet ID, retweet date, original tweet ID and original tweet text. A parser class performs most of the work detecting re-tweets (vs replies and other entries) and extracting out the desired fields.

Three output files are written: "originals.jsonl" (a dictionary of the original tweets), "re_tweets.jsonl" (a dictionary of retweets) and "list_o_r.jsonl" identifies which original tweet is associated with the retweet. The output files anticipate the what we will need to build the graphs - two types of nodes and a set of edges.

Output data format is string and date/time - i.e. no attempt is made yet to make embeddings or transform to torch vectors.

```python
class Parser():
    def __init__(self):

        self.originals = {}
        self.re_tweets = {}
        self.list_o_r = []
        # self.map_o_2_r = {}
        # self.map_r_2_o = {}

    def parse_raw_tweet(self, raw):

        if "retweeted_status" not in raw.keys():
            return

        re_tweet_id = raw["id_str"]
        re_tweet_date = raw["created_at"]
        original_id = raw["retweeted_status"]["id_str"]
        original_text = raw["retweeted_status"]["full_text"]

        self.originals[original_id] = {"text": original_text}
        self.re_tweets[re_tweet_id] = {"date": re_tweet_date}
        self.list_o_r.append((original_id, re_tweet_id))
```

```python
if args.do_parse:

    parser = Parser()

    print("parsing files...")
    for filename in args.parse_file_list:
        print(f" {filename}")
        with open(args.parse_src_folder + filename, "r", encoding="utf-8") as f:
            raw_tweets = ijson.items(f, "", multiple_values=True)

            for raw_tweet in raw_tweets:
                parser.parse_raw_tweet(raw_tweet)

    print("write results...")
    with open(args.parse_dst_folder + "originals.jsonl", "w", encoding="utf-8") as f:
        json.dump(parser.originals, f, indent=4)
    with open(args.parse_dst_folder + "re_tweets.jsonl", "w", encoding="utf-8") as f:
        json.dump(parser.re_tweets, f, indent=4)
    with open(args.parse_dst_folder + "list_o_r.jsonl", "w", encoding="utf-8") as f:
        json.dump(parser.list_o_r, f, indent=4)
```

Filter

A filtering step establishes minimum and maximum threshold on number of retweets. This is to speed up development but also a large percent of tweets only get one or two retweets which is not very interesting. The code (as currently deployed) looks for tweets that receive between 6 and 8 retweets.

Train/Test

The primary tasks are:

- read node and edge data from the preprocessed files
- establish the node and edge lists
- transform any latent data of the nodes/edges into embeddings

Note that inherent node/edge data comes via the nodes themselves - this is different from the neighbor-search embeddings that come later.

Note - our edges do not have data so we do not require edge embeddings.

References:

Sample code:
https://github.com/pyg-team/pytorch_geometric/blob/master/examples/hetero/load_csv.py
https://github.com/pyg-team/pytorch_geometric/blob/master/examples/hetero/*

Tutorial:
https://pytorch-geometric.readthedocs.io/en/latest/tutorial/load_csv.html

https://pytorch-geometric.readthedocs.io/en/latest/tutorial/heterogeneous.html

Non-torch GNN
https://towardsdatascience.com/hands-on-graph-neural-networks-with-pytorch-pytorch-geometric-359487e221a8
https://towardsdatascience.com/a-gentle-introduction-to-graph-neural-network-basics-deepwalk-and-graphsage-db5d540d50b3

A very good top-level view of self-supervised training on graphs is at:
https://medium.com/stanford-cs224w/self-supervised-learning-for-graphs-963e03b9f809

Language Embeddings

A BERT language model used to transform the raw tweet text into an embeddings. Specifically we use the "bert-based-uncased" model from SentenceTransformer at https://www.sbert.net/. The details are as follows:

- The load_nodes_from_file() function takes as input a filename and a list of [column_name: encoder] mappings. The encoder is an instance of SequenceEncoder from https://www.sbert.net/. SequenceEncoder provides the __call__() interface that performs the encoding. SequenceEncoder is instanced within main() at the time of a call to "load_nodes_from_file()".
- load_nodes_from_file() reads the data from the file and proceeds to apply the encoders to the identified data_columns. In our our case, "original_tweet" data has only have one column ("text") and we specify the "bert-based-uncased" SequenceEncoder from https://www.sbert.net/ to be used.

The embeddings step transforms variable-length sentence strings into a fixed-length numeric representation (178 width). The final step of the transforms these into into a torch vector.

```python


from sentence_transformers import SentenceTransformer


class SequenceEncoder:
    r"""'SequenceEncoder' encodes raw column strings into embeddings."""
    def __init__(self, model_name='all-MiniLM-L6-v2', device=None):
        self.device = device
        self.model = SentenceTransformer(model_name, device=device)

    @torch.no_grad()
    def __call__(self, sequence):
        print(f"sequence encoding using {self.model}")
        x = self.model.encode(sequence, show_progress_bar=True,
                              convert_to_tensor=True, device=self.device)
        return x.cpu()



def load_nodes_from_file(filename, encoders=None, **kwargs):

    with open(args.src_folder + filename, "r", encoding="utf-8") as f:
        the_data = json.load(f)

    mapping = {index: i for i, index in enumerate(the_data.keys())}

    x = torch.Tensor()
    if encoders is not None:
        for col, encoder in encoders.items():
            column_of_data = [v[col] for _, v in the_data.items()]
            xs = encoder(column_of_data)
            x = torch.cat((x, xs), dim=-1)

    return x, mapping



def main(args):

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    print("create embeddings for original tweets using BERT-base-uncased")
    original_tweet_x, original_tweet_mapping = load_nodes_from_file(
        filename="originals_filtered.jsonl",
        encoders={
                'text': SequenceEncoder('bert-base-uncased'),
            })
    print(f"original tweets nodes and features: {original_tweet_x.shape}")

```

Date/Time Embeddings

Retweet data has date and time in date/time format data. Here we use a custom encoding that simply maps month and day to nominal values.

```
class DateEncoder:
    r"""DateEncoder encodes month/day columns into embeddings."""
    def __init__(self, device=None):
        self.device = device

    @torch.no_grad()
    def __call__(self, ts):

        monthmap = {
            "Jan": 0,
            "Feb": 1,
            "Mar": 2,
            "Apr": 3,
            "May": 4,
            "Jun": 5,
            "Jul": 6,
            "Aug": 7,
            "Sep": 8,
            "Oct": 9,
            "Nov": 10,
            "Dec": 11,
        }

        source = [row.split(' ') for row in ts]
        embeddings = [[monthmap[row[1]], int(row[2])] for row in source]
        x = torch.Tensor(embeddings)
        return x.cpu()



def main(args):


    retweet_x, retweet_mapping = load_nodes_from_file(
        filename="re_tweets_filtered.jsonl",
        encoders={
                'date': DateEncoder(),
            })
    print(f"retweet nodes and features: {retweet_x.shape}")

    edge_index, edge_label = load_edges_from_file(
        "list_o_r_filtered.jsonl",
        src_index_col=0,
        src_mapping=original_tweet_mapping,
        dst_index_col=1,
        dst_mapping=retweet_mapping,
        encoders=None,
    )
```

Edges

The final component for the graphs are the edges. Our edges do not carry any values/embeddings so it is sufficient to simply list them.

```python


def load_edges_from_file(filename, src_index_col, src_mapping, dst_index_col, dst_mapping,
                         encoders=None, **kwargs):

    with open(args.src_folder + "\\" + filename, "r", encoding="utf-8") as f:
        the_data = json.load(f)

    src = [src_mapping[row[src_index_col]] for row in the_data]
    dst = [dst_mapping[row[dst_index_col]] for row in the_data]
    edge_index = torch.tensor([src, dst])

    edge_attr = None
    # if encoders is not None:
    #     edge_attrs = [encoder(df[col]) for col, encoder in encoders.items()]
    #     edge_attr = torch.cat(edge_attrs, dim=-1)

    return edge_index, edge_attr


def main(args):


    edge_index, edge_label = load_edges_from_file(
        "list_o_r_filtered.jsonl",
        src_index_col=0,
        src_mapping=original_tweet_mapping,
        dst_index_col=1,
        dst_mapping=retweet_mapping,
        encoders=None,
    )

```

Heterogenous Data Object

From above we have two classes of nodes (original_tweet, retweet) and a list of edges. The HeteroData class encapsulate the data providing features such as test/train split, batch-oriented access, sortring and indexing. We create a HeteroData object.

```python
data = HeteroData()
data['original_tweet'].x = original_tweet_x
data['retweet'].x = retweet_x
data['retweet', 'of', 'original_tweet'].edge_index = edge_index
```

Heterogeneous GNN Model

The GNN model is where neighborhood search, aggregation and combining are performed. We use use 2 layers GCNConv and SAGEConv, 64 hidden channels and a linear layer for output. Our optimizer is Adam. We use a standard test/training loop with mse loss.

```python


class GeoCov19HeteroGNN(torch.nn.Module):
    r"""GeoCov19HeteroGNN is a heterogeneous graph based on data from the GeoCov19 Dataset.
    We are following https://pytorch-geometric.readthedocs.io/en/latest/notes/heterogeneous.html#using-the-heteroge
    def __init__(self, hidden_channels, out_channels, num_layers):
        super().__init__()

        self.convs = torch.nn.ModuleList()
        for _ in range(num_layers):
            conv = HeteroConv({
                ('retweet', 'of', 'original_tweet'): GCNConv(-1, hidden_channels, add_self_loops=False),
                ('original_tweet', 'rev_of', 'retweet'): SAGEConv((-1, -1), hidden_channels),
            }, aggr='sum')
            self.convs.append(conv)

        self.lin = Linear(hidden_channels, out_channels)

    def forward(self, x_dict, edge_index_dict):
        for conv in self.convs:
            x_dict = conv(x_dict, edge_index_dict)
            x_dict = {key: x.relu() for key, x in x_dict.items()}
        return self.lin(x_dict['author'])




    model = GeoCov19HeteroGNN(hidden_channels=64,
                              out_channels=3,
                              num_layers=2)
    model = model.to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

Appendix 1: Versions and Platform

The majority of the code is Python with command-line arguments. We used VS-Code as IDE and you will find a launch.json in the .vscode/ folder. To give users something easy to run we added a Colab notebook (referenced in section above).

We used WSL (Windows subsystem for Linux) using Ubuntu 20.04, because there was at least one library that was not supported under Windows. Our host had a NVIDEA GeForce RTX 3060 (12GB) although most of our work was in preprocessing so didn't use the GPU that much.

Conda versions were mostly PyTorch 11.7, Python 3.9.12.

```
# Name                    Version                   Build  Channel
aiohttp                   3.8.4            py39ha55989b_0    conda-forge
aiosignal                 1.3.1              pyhd8ed1ab_0    conda-forge
anyio                     3.5.0            py39haa95532_0
appdirs                   1.4.4              pyh9f0ad1d_0    conda-forge
```

argon2-cffi	20.1.0	py39h2bbff1b_1	anaconda
arrow-cpp	8.0.0	py39hbd6f097_1	
asttokens	2.0.5	pyhd3eb1b0_0	anaconda
async-timeout	4.0.2	pyhd8ed1ab_0	conda-forge
attrs	21.4.0	pyhd3eb1b0_0	anaconda
aws-c-common	0.4.57	ha925a31_1	
aws-c-event-stream	0.1.6	hd77b12b_5	
aws-checksums	0.1.9	ha925a31_0	
aws-sdk-cpp	1.8.185	hd77b12b_0	
babel	2.11.0	py39haa95532_0	
backcall	0.2.0	pyhd3eb1b0_0	anaconda
beautifulsoup4	4.11.1	py39haa95532_0	anaconda
blas	1.0	mkl	
bleach	4.1.0	pyhd3eb1b0_0	anaconda
blosc	1.21.0	h19a0ad4_1	anaconda
boost-cpp	1.78.0	h5b4e17d_0	conda-forge
bottleneck	1.3.5	py39h080aedc_0	
brotli	1.0.9	h2bbff1b_7	
brotli-bin	1.0.9	h2bbff1b_7	
brotlipy	0.7.0	py39h2bbff1b_1003	
bzip2	1.0.8	he774522_0	anaconda
c-ares	1.18.1	h8ffe710_0	conda-forge
ca-certificates	2023.5.7	h56e8100_0	conda-forge
certifi	2023.5.7	pyhd8ed1ab_0	conda-forge
cffi	1.15.1	py39h2bbff1b_3	
cfitsio	3.470	h2bbff1b_7	anaconda
charls	2.2.0	h6c2663c_0	anaconda
charset-normalizer	2.0.4	pyhd3eb1b0_0	
click	8.1.3	win_pyhd8ed1ab_2	conda-forge
cloudpickle	2.0.0	pyhd3eb1b0_0	anaconda
colorama	0.4.4	pyhd3eb1b0_0	anaconda
cryptography	38.0.1	py39h21b164f_0	
cuda	11.7.1	0	nvidia
cuda-cccl	11.7.91	0	nvidia
cuda-command-line-tools	11.7.1	0	nvidia
cuda-compiler	11.7.1	0	nvidia
cuda-cudart	11.7.99	0	nvidia
cuda-cudart-dev	11.7.99	0	nvidia
cuda-cuobjdump	11.7.91	0	nvidia
cuda-cupti	11.7.101	0	nvidia
cuda-cuxxfilt	11.7.91	0	nvidia
cuda-demo-suite	11.8.86	0	nvidia
cuda-documentation	11.8.86	0	nvidia
cuda-libraries	11.7.1	0	nvidia
cuda-libraries-dev	11.7.1	0	nvidia
cuda-memcheck	11.8.86	0	nvidia
cuda-nsight-compute	11.8.0	0	nvidia
cuda-nvcc	11.7.99	0	nvidia
cuda-nvdisasm	11.8.86	0	nvidia
cuda-nvml-dev	11.7.91	0	nvidia
cuda-nvprof	11.8.87	0	nvidia
cuda-nvprune	11.7.91	0	nvidia
cuda-nvrtc	11.7.99	0	nvidia
cuda-nvrtc-dev	11.7.99	0	nvidia
cuda-nvtx	11.7.91	0	nvidia
cuda-nvvp	11.8.87	0	nvidia
cuda-runtime	11.7.1	0	nvidia
cuda-sanitizer-api	11.8.86	0	nvidia
cuda-toolkit	11.7.1	0	nvidia

cuda-tools	11.7.1	0	nvidia
cuda-visual-tools	11.7.1	0	nvidia
cycler	0.11.0	pyhd3eb1b0_0	
cython	0.29.32	py39h99910a6_1	conda-forge
cytoolz	0.11.0	py39h2bbff1b_0	anaconda
dask-core	2022.5.0	py39haa95532_0	anaconda
dataclasses	0.8	pyhc8e2a94_3	conda-forge
datasets	2.10.1	pyhd8ed1ab_0	conda-forge
debugpy	1.5.1	py39hd77b12b_0	anaconda
decorator	5.1.1	pyhd3eb1b0_0	anaconda
defusedxml	0.7.1	pyhd3eb1b0_0	anaconda
dill	0.3.6	pyhd8ed1ab_1	conda-forge
docker-pycreds	0.4.0	py_0	conda-forge
entrypoints	0.4	py39haa95532_0	anaconda
executing	0.8.3	pyhd3eb1b0_0	anaconda
faiss	1.7.2	py39hfccd52d_2_cpu	conda-forge
faiss-cpu	1.7.2	hf9c7e24_2	conda-forge
fftw	3.3.9	h2bbff1b_1	
filelock	3.10.0	pyhd8ed1ab_0	conda-forge
flake8	6.0.0	py39haa95532_0	
flit-core	3.6.0	pyhd3eb1b0_0	
fonttools	4.25.0	pyhd3eb1b0_0	
freetype	2.12.1	ha860e81_0	
frozenlist	1.3.3	py39ha55989b_0	conda-forge
fsspec	2022.3.0	py39haa95532_0	anaconda
gflags	2.2.2	ha925a31_1004	conda-forge
giflib	5.2.1	h62dcd97_0	anaconda
gitdb	4.0.10	pyhd8ed1ab_0	conda-forge
gitpython	3.1.31	pyhd8ed1ab_0	conda-forge
glib	2.69.1	h5dc1a3c_2	
glog	0.6.0	h4797de2_0	conda-forge
gst-plugins-base	1.18.5	h9e645db_0	
gstreamer	1.18.5	hd78058f_0	
huggingface_hub	0.13.3	pyhd8ed1ab_0	conda-forge
icc_rt	2022.1.0	h6049295_2	
icu	58.2	ha925a31_3	
idna	3.4	py39haa95532_0	
ijson	3.2.0.post0	pyhd8ed1ab_0	conda-forge
imagecodecs	2021.8.26	py39hc0a7faf_1	
imageio	2.9.0	pyhd3eb1b0_0	anaconda
importlib-metadata	4.11.3	py39haa95532_0	
importlib_metadata	4.11.3	hd3eb1b0_0	
intel-openmp	2021.4.0	haa95532_3556	
ipykernel	6.9.1	py39haa95532_0	anaconda
ipython	8.3.0	py39haa95532_0	anaconda
ipython_genutils	0.2.0	pyhd3eb1b0_1	anaconda
ipywidgets	7.6.5	pyhd3eb1b0_1	anaconda
jedi	0.18.1	py39haa95532_1	anaconda
jinja2	3.0.3	pyhd3eb1b0_0	anaconda
joblib	1.1.1	py39haa95532_0	
jpeg	9e	h2bbff1b_0	
json5	0.9.6	pyhd3eb1b0_0	
jsonschema	4.4.0	py39haa95532_0	anaconda
jupyter	1.0.0	py39haa95532_8	
jupyter_client	7.2.2	py39haa95532_0	anaconda
jupyter_console	6.4.3	pyhd3eb1b0_0	anaconda
jupyter_core	4.10.0	py39haa95532_0	anaconda
jupyter_server	1.23.4	py39haa95532_0	
jupyterlab	3.5.3	py39haa95532_0	

```
jupyterlab_pygments       0.1.2                      py_0    anaconda
jupyterlab_server         2.16.5           py39haa95532_0
jupyterlab_widgets        1.0.0              pyhd3eb1b0_1    anaconda
kiwisolver                1.4.2            py39hd77b12b_0
lcms2                     2.12                 h83e58a3_0    anaconda
lerc                      3.0                  hd77b12b_0
libaec                    1.0.4                h33f27b4_1    anaconda
libblas                   3.9.0           1_h8933c1f_netlib    conda-forge
libbrotlicommon           1.0.9                h2bbff1b_7
libbrotlidec              1.0.9                h2bbff1b_7
libbrotlienc              1.0.9                h2bbff1b_7
libclang                  12.0.0          default_h627e005_2
libcublas                 11.11.3.6                     0    nvidia
libcublas-dev             11.11.3.6                     0    nvidia
libcufft                  10.9.0.58                     0    nvidia
libcufft-dev              10.9.0.58                     0    nvidia
libcurand                 10.3.0.86                     0    nvidia
libcurand-dev             10.3.0.86                     0    nvidia
libcurl                   7.88.1               h86230a5_0
libcusolver               11.4.1.48                     0    nvidia
libcusolver-dev           11.4.1.48                     0    nvidia
libcusparse               11.7.5.86                     0    nvidia
libcusparse-dev           11.7.5.86                     0    nvidia
libdeflate                1.8                  h2bbff1b_5
libfaiss                  1.7.2             hba6d9cf_2_cpu    conda-forge
libfaiss-avx2             1.7.2             h1234567_2_cpu    conda-forge
libffi                    3.4.2                hd77b12b_6
libiconv                  1.16                 h2bbff1b_2
liblapack                 3.9.0           5_hd5c7e75_netlib    conda-forge
libnpp                    11.8.0.86                     0    nvidia
libnpp-dev                11.8.0.86                     0    nvidia
libnvjpeg                 11.9.0.86                     0    nvidia
libnvjpeg-dev             11.9.0.86                     0    nvidia
libogg                    1.3.5                h2bbff1b_1
libpng                    1.6.37               h2a8f88b_0
libprotobuf               3.20.3               h23ce68f_0
libssh2                   1.10.0               h680486a_2    conda-forge
libthrift                 0.15.0               h636ae23_0    conda-forge
libtiff                   4.4.0                h8a3f274_2
libuv                     1.40.0               he774522_0
libvorbis                 1.3.7                he774522_0
libwebp                   1.2.4                h2bbff1b_0
libwebp-base              1.2.4                h2bbff1b_0
libxml2                   2.9.14               h0ad7f3c_0
libxslt                   1.1.35               h2bbff1b_0
libzopfli                 1.0.3                ha925a31_0    anaconda
locket                    1.0.0            py39haa95532_0    anaconda
lxml                      4.9.1            py39h1985fb9_0
lz4-c                     1.9.4                h2bbff1b_0
m2w64-gcc-libgfortran     5.3.0                         6    conda-forge
m2w64-gcc-libs            5.3.0                         7    conda-forge
m2w64-gcc-libs-core       5.3.0                         7    conda-forge
m2w64-gmp                 6.1.0                         2    conda-forge
m2w64-libwinpthread-git   5.0.0.4634.697f757               2    conda-forge
markupsafe                2.1.1            py39h2bbff1b_0    anaconda
matplotlib                3.5.3            py39haa95532_0
matplotlib-base           3.5.3            py39hd77b12b_0
matplotlib-inline         0.1.2              pyhd3eb1b0_2    anaconda
mccabe                    0.7.0              pyhd3eb1b0_0
```

mistune	0.8.4	py39h2bbff1b_1000	anaconda
mkl	2021.4.0	haa95532_640	
mkl-service	2.4.0	py39h2bbff1b_0	
mkl_fft	1.3.1	py39h277e83a_0	
mkl_random	1.2.2	py39hf11a4ad_0	
msys2-conda-epoch	20160418	1	conda-forge
multidict	6.0.4	py39ha55989b_0	conda-forge
multiprocess	0.70.14	py39ha55989b_3	conda-forge
munkres	1.1.4	py_0	
nbclassic	0.5.2	py39haa95532_0	
nbclient	0.5.13	py39haa95532_0	anaconda
nbconvert	6.4.4	py39haa95532_0	anaconda
nbformat	5.3.0	py39haa95532_0	anaconda
nest-asyncio	1.5.5	py39haa95532_0	anaconda
networkx	2.7.1	pyhd3eb1b0_0	anaconda
nltk	3.8.1	pyhd8ed1ab_0	conda-forge
notebook	6.4.11	py39haa95532_0	anaconda
notebook-shim	0.2.2	py39haa95532_0	
nsight-compute	2022.3.0.22	0	nvidia
numexpr	2.8.4	py39h5b0cc5e_0	
numpy	1.23.4	py39h3b20f71_0	
numpy-base	1.23.4	py39h4da318b_0	
openjpeg	2.4.0	h4fc8c34_0	anaconda
openssl	1.1.1t	hcfcfb64_0	conda-forge
opt_einsum	3.3.0	pyhd8ed1ab_1	conda-forge
packaging	21.3	pyhd3eb1b0_0	
pandas	1.5.1	py39hf11a4ad_0	
pandocfilters	1.5.0	pyhd3eb1b0_0	anaconda
parso	0.8.3	pyhd3eb1b0_0	anaconda
partd	1.2.0	pyhd3eb1b0_1	anaconda
pathtools	0.1.2	py_1	conda-forge
pcre	8.45	hd77b12b_0	
pickleshare	0.7.5	pyhd3eb1b0_1003	anaconda
pillow	9.2.0	py39hdc2b20a_1	
pip	22.3.1	py39haa95532_0	
ply	3.11	py39haa95532_0	
portalocker	2.7.0	py39hcbf5309_0	conda-forge
prometheus_client	0.13.1	pyhd3eb1b0_0	anaconda
prompt-toolkit	3.0.20	pyhd3eb1b0_0	anaconda
prompt_toolkit	3.0.20	hd3eb1b0_0	anaconda
protobuf	3.20.3	py39hd77b12b_0	
psutil	5.9.4	py39ha55989b_0	conda-forge
pure_eval	0.2.2	pyhd3eb1b0_0	anaconda
pyarrow	8.0.0	py39h953b917_0	
pycocotools	2.0	pypi_0	pypi
pycodestyle	2.10.0	py39haa95532_0	
pycparser	2.21	pyhd3eb1b0_0	
pyflakes	3.0.1	py39haa95532_0	
pyg	2.3.0	py39_torch_1.13.0_cu117	pyg
pygments	2.11.2	pyhd3eb1b0_0	anaconda
pynvml	11.4.1	pyhd8ed1ab_0	conda-forge
pyopenssl	22.0.0	pyhd3eb1b0_0	
pyparsing	3.0.9	py39haa95532_0	
pyqt	5.15.7	py39hd77b12b_0	
pyqt5-sip	12.11.0	py39hd77b12b_0	
pyro-api	0.1.2	pyhd8ed1ab_0	conda-forge
pyro-ppl	1.8.4	pyhd8ed1ab_0	conda-forge
pyrsistent	0.18.0	py39h196d8e1_0	anaconda
pysocks	1.7.1	py39haa95532_0	

python	3.9.15	h6244533_2	
python-dateutil	2.8.2	pyhd3eb1b0_0	
python-fastjsonschema	2.15.1	pyhd3eb1b0_0	anaconda
python-xxhash	3.2.0	py39ha55989b_0	conda-forge
python_abi	3.9	2_cp39	conda-forge
pytorch	1.13.0	py3.9_cuda11.7_cudnn8_0	pytorch
pytorch-cuda	11.7	h67b0de4_0	pytorch
pytorch-mutex	1.0	cuda	pytorch
pytz	2022.1	py39haa95532_0	
pywavelets	1.3.0	py39h2bbff1b_0	anaconda
pywin32	302	py39h2bbff1b_2	anaconda
pywinpty	2.0.2	py39h5da7b33_0	anaconda
pyyaml	6.0	py39h2bbff1b_1	anaconda
pyzmq	22.3.0	py39hd77b12b_2	anaconda
qt-main	5.15.2	he8e5bd7_7	
qt-webengine	5.15.9	hb9a9bb5_4	
qtconsole	5.3.0	pyhd3eb1b0_0	anaconda
qtpy	2.0.1	pyhd3eb1b0_0	anaconda
qtwebkit	5.212	h3ad3cdb_4	
re2	2022.04.01	h0e60522_0	conda-forge
regex	2022.10.31	py39ha55989b_0	conda-forge
requests	2.28.1	py39haa95532_0	
responses	0.18.0	pyhd8ed1ab_0	conda-forge
sacrebleu	2.3.1	pyhd8ed1ab_0	conda-forge
sacremoses	0.0.53	pyhd8ed1ab_0	conda-forge
scikit-image	0.19.2	py39hf11a4ad_0	anaconda
scikit-learn	1.1.3	py39hd77b12b_0	
scipy	1.9.3	py39he11b74f_0	
send2trash	1.8.0	pyhd3eb1b0_1	anaconda
sentence-transformers	2.2.2	pyhd8ed1ab_0	conda-forge
sentencepiece	0.1.95	py39h59b6b97_0	
sentry-sdk	1.16.0	pyhd8ed1ab_0	conda-forge
setproctitle	1.3.2	py39ha55989b_1	conda-forge
setuptools	65.5.0	py39haa95532_0	
sip	6.6.2	py39hd77b12b_0	
six	1.16.0	pyhd3eb1b0_1	
smmap	3.0.5	pyh44b312d_0	conda-forge
snappy	1.1.9	h6c2663c_0	anaconda
sniffio	1.2.0	py39haa95532_1	
soupsieve	2.3.1	pyhd3eb1b0_0	anaconda
sqlite	3.40.0	h2bbff1b_0	
stack_data	0.2.0	pyhd3eb1b0_0	anaconda
tabulate	0.9.0	pyhd8ed1ab_1	conda-forge
terminado	0.13.1	py39haa95532_0	anaconda
testpath	0.6.0	py39haa95532_0	anaconda
threadpoolctl	2.2.0	pyh0d69192_0	
tifffile	2021.7.2	pyhd3eb1b0_2	anaconda
tk	8.6.12	h2bbff1b_0	
tokenizers	0.13.2	py39hca44cb7_0	conda-forge
toml	0.10.2	pyhd3eb1b0_0	
tomli	2.0.1	py39haa95532_0	
toolz	0.11.2	pyhd3eb1b0_0	anaconda
torchaudio	0.13.0	pypi_0	pypi
torchinfo	1.7.1	pyhd8ed1ab_0	conda-forge
torchvision	0.14.0	pypi_0	pypi
tornado	6.2	py39h2bbff1b_0	
tqdm	4.65.0	pyhd8ed1ab_1	conda-forge
traitlets	5.1.1	pyhd3eb1b0_0	anaconda
transformers	4.27.1	pyhd8ed1ab_0	conda-forge

typing	3.10.0.0	pyhd8ed1ab_0	conda-forge
typing-extensions	4.4.0	py39haa95532_0	
typing_extensions	4.4.0	py39haa95532_0	
tzdata	2022g	h04d1e81_0	
ucrt	10.0.20348.0	haa95532_0	
urllib3	1.26.13	py39haa95532_0	
utf8proc	2.6.1	h2bbff1b_0	
vc	14.2	h21ff451_1	
vs2015_runtime	14.32.31332	h1d6e394_9	conda-forge
wandb	0.14.0	pyhd8ed1ab_0	conda-forge
wcwidth	0.2.5	pyhd3eb1b0_0	anaconda
webencodings	0.5.1	py39haa95532_1	anaconda
websocket-client	0.58.0	py39haa95532_4	
wget	3.2	pypi_0	pypi
wheel	0.37.1	pyhd3eb1b0_0	
widgetsnbextension	3.5.2	py39haa95532_0	anaconda
win_inet_pton	1.1.0	py39haa95532_0	
wincertstore	0.2	py39haa95532_2	
winpty	0.4.3	4	anaconda
wordcloud	1.8.2.2	py39ha55989b_1	conda-forge
xxhash	0.8.1	hcfcfb64_0	conda-forge
xz	5.2.8	h8cc25b3_0	
yaml	0.2.5	he774522_0	anaconda
yarl	1.8.2	py39ha55989b_0	conda-forge
zfp	0.5.5	hd77b12b_6	anaconda
zipp	3.11.0	py39haa95532_0	
zlib	1.2.13	h8cc25b3_0	
zstd	1.5.2	h19a0ad4_0	