Code Explanation: Single Dataset Graph Preprocessing

Author Name

March 31, 2025

1 Introduction

This document describes a Python script designed to preprocess metadata for a single dataset, build relationships among articles (based on shared authors, year, and journal), and finally construct a directed graph stored on disk. The script requires:

1. The number of threads (to parallelize certain computations).
2. The path to a metadata dataset (e.g., `.hf` files loaded via `datasets.load_from_disk`).
3. The path to a retrieval dataset containing indices that refer to relevant metadata entries.

By combining these inputs, the script identifies which articles need to be processed (using the `retrieval_relative_indices` field) and establishes metadata-based relationships such as:

- `same_year`
- `same_journal`
- `author_in_common`
- `chemical_term_in_common`
- `mesh_term_in_common`

Once those relationships are computed, it constructs a graph with `networkx.MultiDiGraph` and saves it as a `pickle` file.

2 Script Overview

The Python code starts by importing the required libraries, then defines several utility functions that handle:

- Conversion of dictionaries of lists into lists of dictionaries,
- Flattening lists of lists,
- Filtering out invalid (e.g. `-1`) indices,
- Parallel computation of relationships,
- Building and saving the final graph.

Finally, in the `if __name__ == "__main__":` block, the program prompts the user to provide the three required command-line parameters (or standard input if used interactively), processes the data, and writes out both a JSON file containing the list of relationships and a `.pickle` file containing the constructed graph.

3 Key Code Snippets

3.1 Parallel Relationship Creation

Below is a brief snippet demonstrating how the script processes each index in parallel. Each task calls `process_relation_index`, which checks for matches (e.g., same year) between articles.

Listing 1: Parallel creation of metadata relationships

```python
def create_metadata_relations_parallel(fname, db_metadata, retr_info_ds,
    relation_handler, num_workers):
    indices = retr_info_ds[fname]["retrieval_relative_indices"]
    all_relations = []
    with ProcessPoolExecutor(max_workers=num_workers) as executor:
        futures = [
            executor.submit(
                process_relation_index,
                i, fname, db_metadata, retr_info_ds, relation_handler
            )
            for i in range(len(indices))
        ]
        for future in tqdm(as_completed(futures), total=len(indices),
                           desc="Parallel creating metadata relations"):
            try:
                all_relations.extend(future.result())
            except Exception as e:
                print(f"Error in processing index: {e}")
    return all_relations
```

3.2 Constructing the Graph

Once all relationships are gathered, the script builds a directed graph with NetworkX. We map article IDs (e.g., `PMID`) to integer nodes and add edges that represent each relationship.

Listing 2: Constructing and saving the metadata graph

```
def construct_metadata_graph(article_pmids, metadata_relations,
                             output_path, relation_handler):
    concept2id = {pmid: i for i, pmid in enumerate(article_pmids)}
    graph = nx.MultiDiGraph()

    for (subj_pmid, obj_pmid, rel_id) in metadata_relations:
        try:
            s = concept2id[subj_pmid]
            o = concept2id[obj_pmid]
        except KeyError:
            continue
        graph.add_edge(s, o, rel=rel_id, weight=1.0)
        graph.add_edge(o, s, rel=rel_id, weight=1.0)

    with open(output_path, "wb") as f:
        pickle.dump(graph, f)
    return graph
```

4 Execution and Usage

1. Make the script executable:

   ```
   chmod +x graph_preprocessing_script.py
   ```

2. Run it in a shell or terminal, providing the required arguments and following interactive prompts:

   ```
   ./graph_preprocessing_script.py
   ```

3. Provide the number of threads and valid paths to both the metadata dataset and the retrieval dataset when prompted.
4. The script will create:
    - `metadata_relations.json`, containing all discovered relationships.
    - `metadata_graph.pickle`, containing the final NetworkX graph object.

5 Conclusion

This script is a basic yet functional approach to building a graph of relationships among articles based on metadata such as shared authors, journal, or publication year. By exploiting `ProcessPoolExecutor`, it scales efficiently to large datasets and provides both a human-readable JSON file of relationships and a `pickle`-based graph for downstream tasks.