CS 768 Assignment

Aditya Singh
22b0056@iitb.ac.in

The code for this assignment is available at https://github.com/adityasz/cs768-assignment. Detailed instructions for setup and running the scripts, as well as reproducing all the results including this report (with all the figures and tables), are available in the README.

Task 1: Build a citation graph

`*.bbl` files do not follow a standard spec. It is impossible to parse them reliably. A large number of the `*.bib` files were also "broken". So, I first tried to use Semantic Scholar's Academic Graph API – the code for this is in `scripts/create_dataset_semantic_scholar.py`.

They have a huge dataset that contains the references of essentially every paper. They block IITB LAN (at least the public IP of Hostel 3 is blocked), but Eduroam works (I am responsible and added sufficient time delay between calls so as to not get rate limited: The block is due to someone else). They also support batched API calls: With a batch of 300 papers, only ~ 22 API calls (5 seconds apart) are enough to fetch everything. The only problem is that their references lists are incomplete, and it identifies only $\sim 18\,000$ edges in the given dataset.

OpenAlex also claims to have a graph database. Their `Work` object has a `referenced_works` list that consists of citations going *from* one work *to* another work. The problem here is that `referenced_works` is empty for every paper I looked up.

So, I had to manually parse `.bbl` files. I did the following:

1. Make everything lowercase.
2. Remove all characters not in the class `[^ a-z0-9\n\t]`.
3. Remove `bititem` and `newblock` from all `*.bbl` files.
4. For `bib` files that are not broken, just extract the titles.
5. Combine the preprocessed data form all the `.bib` and `.bbl` files of each paper into one file named `super_simple_refs.txt`.

Now, for each arXiv ID, I fuzzy searched every other arXiv ID in its `super_simple_refs.txt`. `rapidfuzz.fuzz.partial_ratio` with a threshold of 95 was used. Search was done in parallel across 6 cores (laptops have thermal constraints and they can only run a small number of cores at a high frequency – using more cores does not help beyond a certain amount, depending on the task and the heat sink's heat dissipation capacity).

The code for this is in `scripts/preprocess_dataset.py`.

The data was preprocessed into a `dict[arXivId, Paper]`, where `arXivId = str` and `Paper` is a with the following structure:

```python
from dataclasses import dataclass


@dataclass
class Paper:
    title: str
    """The title of the paper."""
    abstract: str
    """The abstract of the paper."""
    references: set[arXivId]
    """The arXiv IDs of the papers that this paper cites."""
```

The dataset is stored as a gzipped JSON file. NetworkX was then used to build a directed graph based on the references. The code for this is in `scripts/analyze_graph.py`. The statistics are in Table 1. The (log-scale) histograms of the in- and out-degrees are in Figure 2 and Figure 3.

Number of edges	28 254
Number of isolated nodes	463
Average node degree	4.32
Diameter	3

Table 1: Some statistics of the citation graph.

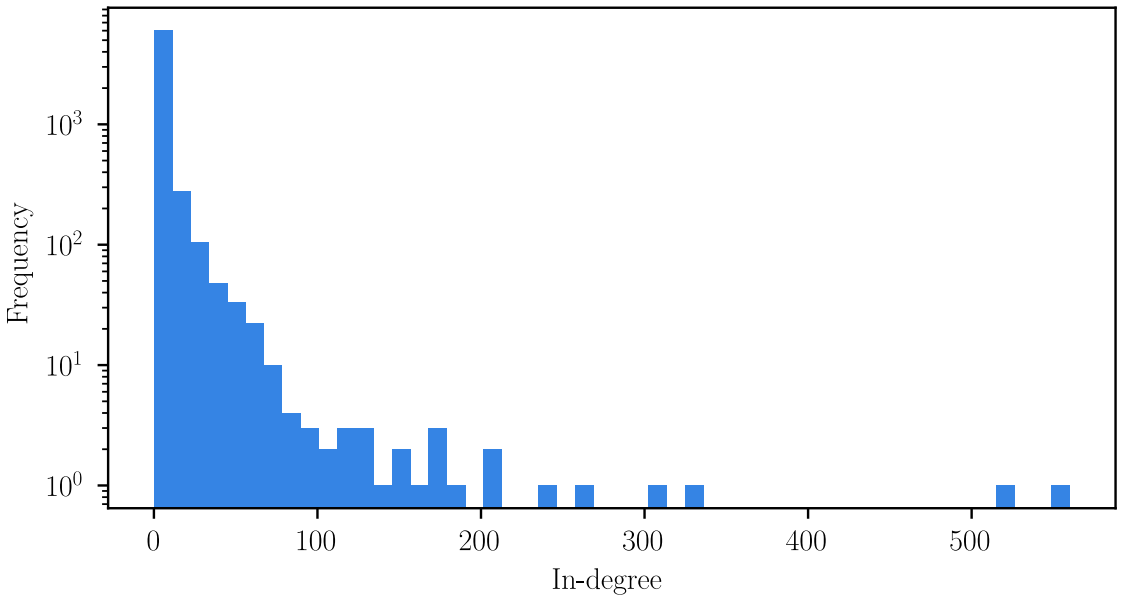

Figure 2: The distribution of the in-degree of the nodes in the citation graph.

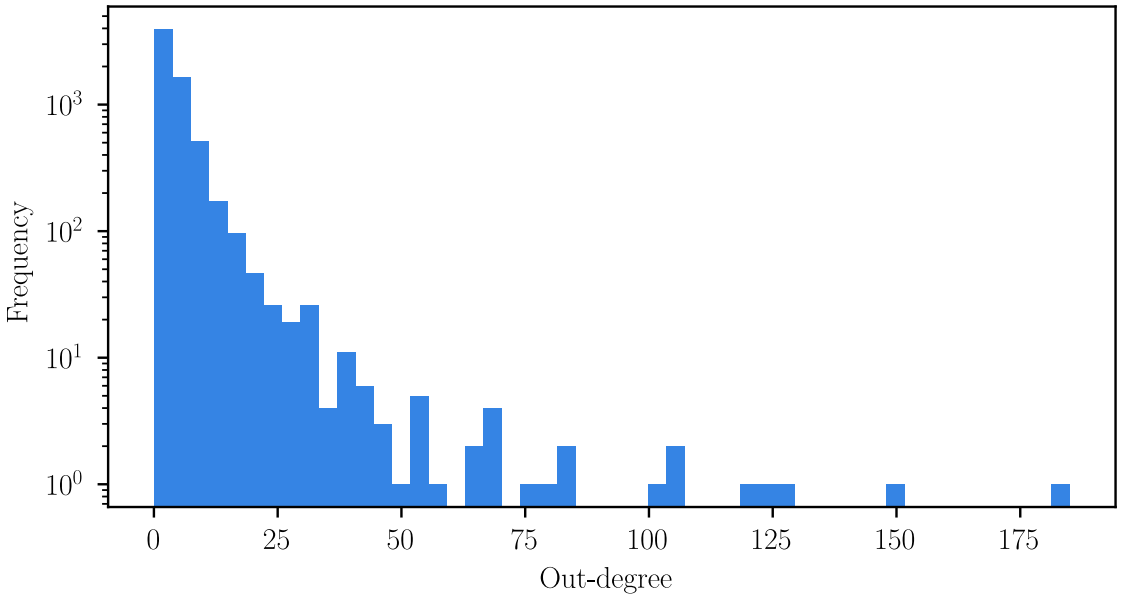

Figure 3: The distribution of the out-degree of the nodes in the citation graph.

Task 2: Machine learning

For citation prediction, I looked up several papers to get an idea of how to approach this problem. There were several different approaches, and [1] achieves good results. They first do a nearest neighbor search based on the embeddings of the papers (prefetching stage), and then use a SciBERT based model to rerank the initial list. However, they also utilized local citation context, and that is not available in this assignment's dataset. Also, this assignment's dataset is very small – it contains only titles and abstracts, and training the network in [1] wouldn't give good results.

I tried training different GNNs, but to train a large transformer based model to get initial features from the titles and abstracts requires a large dataset. Also, [1] itself uses SciBERT for reranking. So, I compared several open source models from hugging face to get the initial features and found SPECTER2 [2] to be good (no rigorous testing was done, just a handful of comparisons on the given dataset were made). I then generated the initial embeddings and trained a GATv2-like network [3]. I trained it with different positive- and negative-edge sampling ratios, but the recall at K was very low. Adding heuristics to rerank (like most cited among top-N, where I would hope that the unknown $K > N$, etc.) did not help.

Simply returning the top-K papers based on the cosine similarities of their embeddings with the query paper resulted in a recall at 50 of ~ 0.38. I could not beat this with heuristics or GNN – though I did not have much time to try because of other assignments and time constraints (maybe the GNN had a bug in the sampling).

The code to generate embeddings is available in `scripts/generate_embeddings.py`.

Bibliography

[1] N. Gu, Y. Gao, and R. H. R. Hahnloser, "Local Citation Recommendation with Hierarchical-Attention Text Encoder and SciBERT-based Reranking," *CoRR*, 2021, [Online]. Available: https://arxiv.org/abs/2112.01206

[2] A. Singh, M. D'Arcy, A. Cohan, D. Downey, and S. Feldman, "SciRepEval: A Multi-Format Benchmark for Scientific Document Representations," in *Conference on Empirical Methods in Natural Language Processing*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254018137

[3] S. Brody, U. Alon, and E. Yahav, "How Attentive are Graph Attention Networks?," *CoRR*, 2021, [Online]. Available: https://arxiv.org/abs/2105.14491