Tec Narayan Brahmachari (22b0982)
Navya (22b1007)
Adwai (22b1282)
Sambhav (22b0932)

May 5, 2025

`https://github.com/tecnarayan/cs768-2025-assignment`

1 Introduction

This report details the approach and results for the CS768 assignment focused on analyzing a citation network and predicting potential citations. The assignment involves two main tasks using a dataset of approximately 6500 research papers from NeurIPS and ICML conferences. The dataset for each paper includes its title, abstract, and bibliography files (`.bbl` and/or `.bib`).

Task 1 required building a citation graph from the provided dataset and analyzing its properties. **Task 2** involved developing a machine learning model to predict which papers from the dataset a new, unseen paper might cite, evaluated using Recall@K.

2 Task 1: Citation Graph Construction and Analysis

2.1 Graph Construction Method

A directed citation graph $G = (V, E)$ was constructed, where V is the set of papers in the dataset (identified by their folder names/IDs) and a directed edge $(u, v) \in E$ exists if paper u cites paper v.

The process involved:

1. **Data Extraction:** For each paper u, the title was read from `title.txt` and potential citation reference strings were extracted from its `.bbl` and `.bib` files. Specifically, from `.bbl` files, text following the first `\newblock` in each `\bibitem` was extracted. From `.bib` files, the content of the `title` field was extracted.
2. **Text Normalization:** A crucial step was normalizing both the extracted citation strings and the titles of all papers in the dataset to handle variations in formatting, casing, punctuation, and common extraneous information. The final normalization function (`normalize_text` in the code) performs the following:
   - Converts text to lowercase.
   - Separates known joined common ML terms (e.g., `deeplearning` to `deep learning`) based on a predefined dictionary (`known_joins`).
   - Replaces hyphens between word characters with spaces.
   - Removes content within LaTeX curly braces ({...}).
   - Removes all punctuation characters except whitespace.
   - Normalizes whitespace (multiple spaces/newlines to single spaces, trims ends).
3. **Lookup Table Creation:** A dictionary was built mapping each unique normalized title found in the dataset to the list of paper IDs having that title. This allows for efficient matching.

4. **Edge Creation:** For each paper u and each of its normalized citation strings c_{norm}, the lookup table was queried. If c_{norm} matched a normalized title associated with paper ID v (where $u \neq v$), a directed edge (u, v) was added to the graph.

This process was implemented in `citation_graph_generator.py`.

2.2 Graph Properties

The constructed directed graph using the final normalization method yielded the following properties:

- Number of Nodes (Papers): 6545
- Number of Edges (Citations Found): 30762
- Number of Isolated Nodes: 441
- Average In-Degree: 4.7001
- Average Out-Degree: 4.7001
- Diameter of Largest Weakly Connected Component (LWCC): 13 (The LWCC contains 6055 nodes).

2.3 Degree Distribution

The distribution of node degrees (total degree = in-degree + out-degree) provides insight into the connectivity patterns.

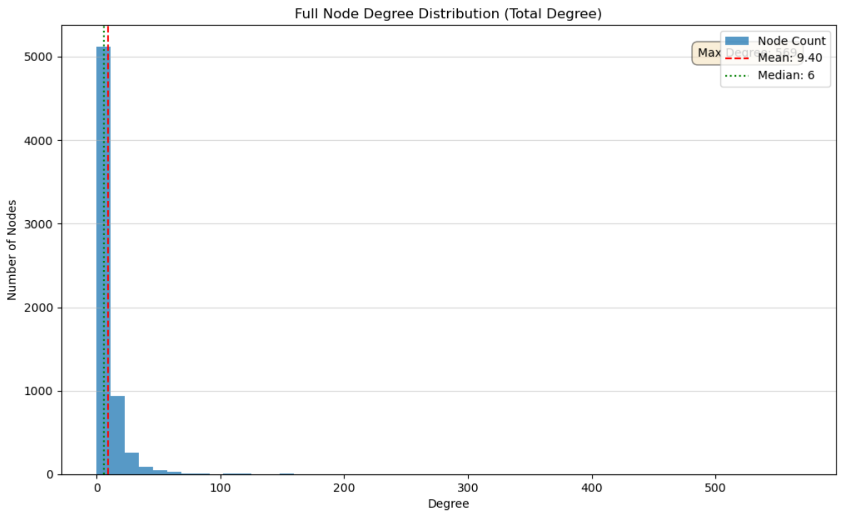

Figure 1: Distribution of Total Node Degrees for the Full Graph. Vertical lines indicate Mean (red dashed) and Median (green dotted) degrees.

Key statistics for the total degree distribution:

- Average Total Degree: 9.4002
- Median Total Degree: 6

The histogram (Figure 1) shows a highly skewed distribution, typical for real-world networks, with many nodes having low degrees and a few nodes having very high degrees (hubs).

To better visualize the bulk of the distribution, an analysis was performed on a filtered graph, removing nodes in the bottom 5% and top 5% based on total degree.

- Degree Thresholds Used for Filtering: ≥ 0.0 and ≤ 28.0
- Nodes Remaining After Filtering: 6232 (out of 6545)
- Edges Remaining After Filtering: 14013
- Filtered Average Total Degree: 4.4971
- Filtered Median Total Degree: 3.0

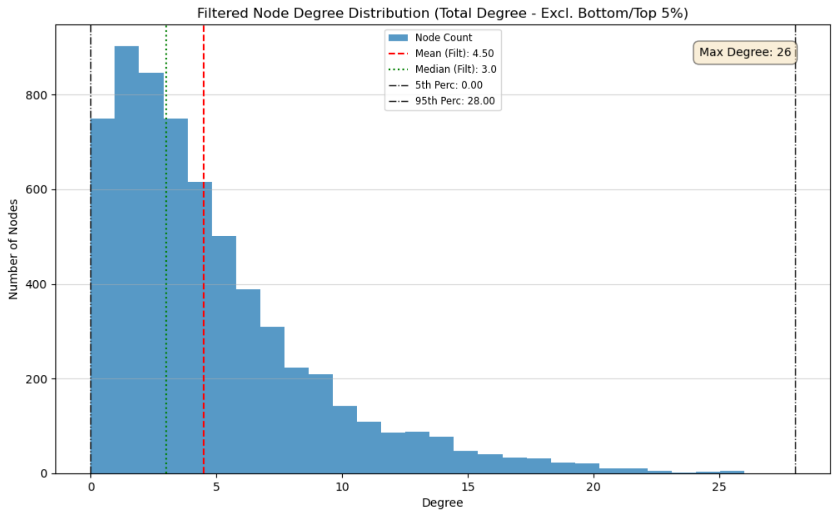

Figure 2: Distribution of Total Node Degrees (Filtered Graph: Bottom/Top 5% Removed). Vertical lines indicate Mean (red dashed), Median (green dotted), and the 5th/95th percentile degree thresholds (black dash-dot) used for filtering.

The filtered histogram (Figure 2) provides a clearer view of the distribution for the majority of nodes, excluding the extreme outliers.

2.4 In-Degree Analysis

The in-degree of a node represents the number of citations it received from other papers within the dataset, often indicating influence or popularity. This analysis was performed by `in_degree.py`.

Key statistics for the in-degree distribution:

- Mean In-Degree: 4.7001 (Matches graph average, as expected)
- Median In-Degree: 1
- Max In-Degree: 617

Top 5 Most Cited Papers (Highest In-Degree):

1. Paper ID: 1912.01703v1, In-Degree: 568
2. Paper ID: 2005.14165v4, In-Degree: 550
3. Paper ID: 1502.03167v3, In-Degree: 503
4. Paper ID: 1703.03400v3, In-Degree: 306

5. Paper ID: 2103.00020v1, In-Degree: 293

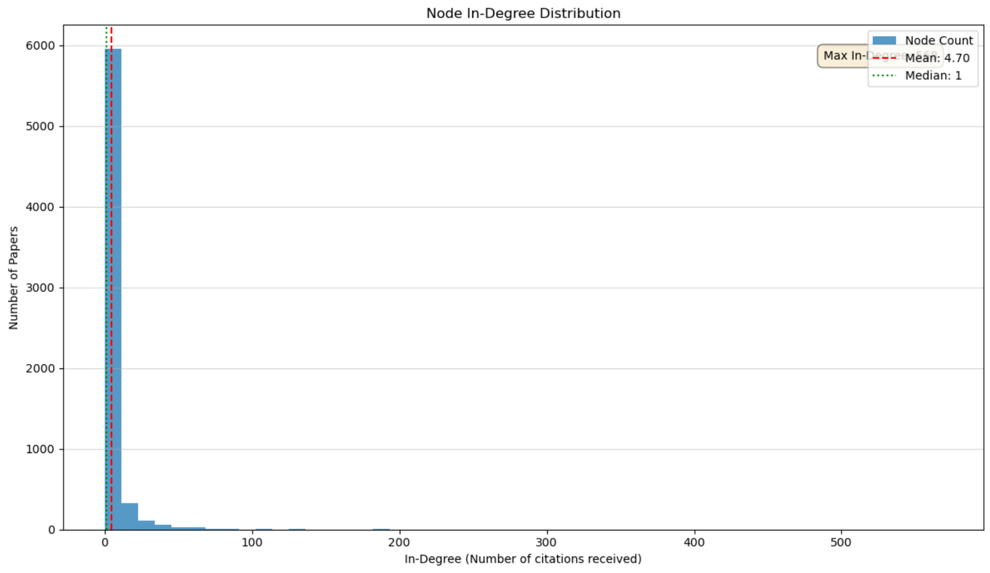

Figure 3: Distribution of Node In-Degrees. Vertical lines indicate Mean (red dashed) and Median (green dotted) in-degrees.

The in-degree distribution (Figure 3) is also highly skewed, with most papers receiving very few citations (median is 1) and a small number receiving a large number of citations.

3 Task 2: Link Prediction

3.1 Approach Overview

The goal of Task 2 was to predict, given the title and abstract of a new paper, a ranked list of papers from the dataset that the new paper might cite. A hybrid approach combining content similarity (TF-IDF) and graph-based node importance (In-Degree) was implemented in `evaluation.py`. The approach uses a two-stage ranking strategy based on candidate selection.

3.2 Offline Precomputation

To ensure efficient online prediction, several components were precomputed using `precompute_data.py`:

1. **Text Preparation:** The title and abstract for each of the 6545 dataset papers were concatenated.
2. **TF-IDF Model Training:** A `TfidfVectorizer` was trained on the concatenated, normalized texts using unigrams (single words) and English stop words. The vocabulary size was 24831. The fitted vectorizer (containing vocabulary and IDF scores) was saved to `tfidf_vectorizer.pkl`.
3. **Dataset Vectors:** The TF-IDF vectors for all 6545 papers were computed using the fitted vectorizer and saved as a sparse matrix to `dataset_tfidf_vectors.npz`.
4. **ID Ordering:** The order of paper IDs corresponding to the rows of the TF-IDF matrix was saved to `paper_id_order.txt`.
5. **In-Degrees:** The in-degree for each paper was precomputed using `in_degree.py` and saved to `in_degrees.txt`. The maximum in-degree (MaxID) was noted for normalization.

3.3 Online Prediction (`evaluation.py`)

When `evaluation.py` receives a new paper's title and abstract:

1. **Load Data:** The precomputed vectorizer, dataset vectors, paper IDs, in-degrees, max in-degree, and the citation graph are loaded once (using a cache).

2. **Input Processing:** The input title and abstract are combined and normalized using the standard `normalize_text` function.

3. **Input Vectorization:** The normalized input text is transformed into a TF-IDF vector using the loaded vectorizer. Words not present in the original vocabulary are ignored.

4. **Content Similarity:** Cosine similarity is calculated between the input vector and all precomputed dataset vectors, yielding `content_scores` for each dataset paper.

5. **Seed Selection:** Papers are sorted by `content_scores`. The top `SEED_COUNT` papers (a hyperparameter, set to 5).

6. **Candidate Expansion:** A candidate set `candidate_set_M` is initialized with the `seed_nodes`. For each seed node, its direct successors (papers it cites) from the loaded graph are added to `candidate_set_M`.

7. **Candidate Ranking (Stage 1):** For each paper `p` in `candidate_set_M`, a hybrid score is calculated:

$$\text{Score}_{\text{final}}(p) = \alpha \times \text{Score}_{\text{TF-IDF}}(p) + (1-\alpha) \times \frac{\text{InDegree}(p)}{\text{MaxID}_{\text{eff}}}$$

   where `Score_TF-IDF` is the cosine similarity, `InDegree(p)` is the precomputed in-degree, `MaxID_eff` is the effective maximum in-degree used for normalization (set to 50 in the code, although the actual max was higher), and α is a hyperparameter (set to 0.7). The candidates in `M` are ranked by this score (`ranked_candidates`).

8. **Remaining Ranking (Stage 2):** Papers from the full dataset that were *not* in `candidate_set_M` are identified (`remaining_paper_ids_list`). A score is calculated for these using a different weight, β:

$$\text{Score}_{\text{remaining}}(p) = \beta \times \text{Score}_{\text{TF-IDF}}(p) + (1-\beta) \times \frac{\text{InDegree}(p)}{\text{MaxID}_{\text{eff}}}$$

   where β was set to 0.9. These remaining papers are ranked by this score (`ranked_remaining`).

9. **Final Output:** The final ranked list `result` is produced by concatenating `ranked_candidates` and `ranked_remaining`. This full list is printed to standard output, one paper ID per line.

Note on Max In-Degree: The code currently uses a hardcoded `MAX_IN_DEGREE` of 50 for normalization.

3.4 Evaluation Simulation

An `autograder.py` script was developed to simulate the evaluation process. It randomly samples N papers from the dataset, reads their title/abstract, and feeds them to `evaluation.py`. It retrieves the ranked list of predictions and compares the top K predictions against the ground truth citations (successors) for the test paper in the pre-built `citation_graph.adjlist`.

The metric calculated is the **Average Recall@K**, defined as the average, over all tested papers with at least one actual citation, of the fraction of their actual citations found within the top K predictions. We got recall 0.35 for K $\bar{1}0$, Note you have to modify evaluation.py to make sure you don't incude the same file as seed.

4 Content Similarity Analysis

To understand the typical content similarity between citing and cited papers, an analysis was performed (`analysis/similarity_score.py`). For each edge (u, v) in the citation graph, the cosine similarity between the TF-IDF vectors (derived from title+abstract) of paper u and paper v was calculated.

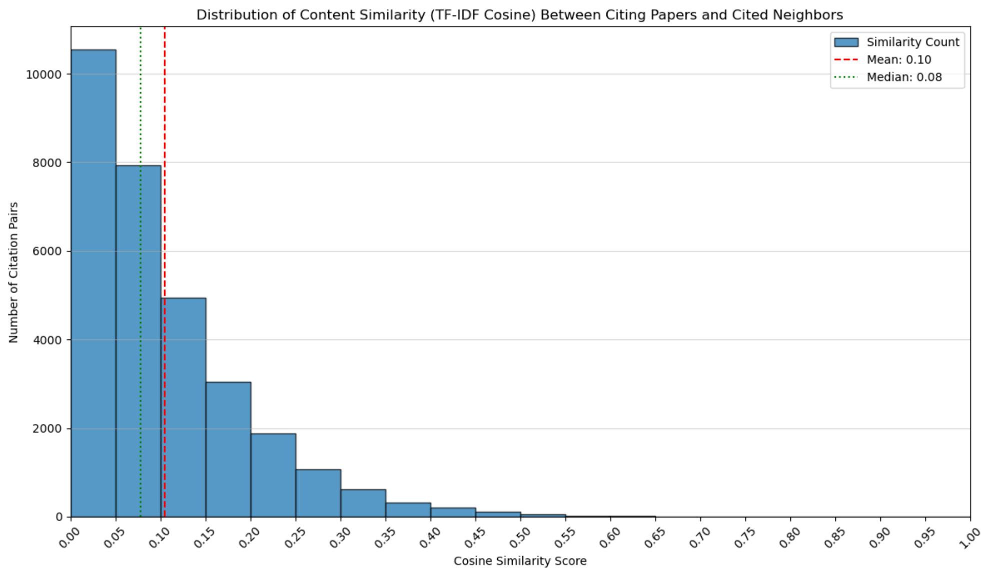

Figure 4: Distribution of Cosine Similarity between Citing and Cited Papers.

Results for the 30762 citation pairs found:

- Mean Similarity: 0.1046
- Median Similarity: 0.0781
- Mode Similarity (rounded to 0.01): 0.02

The histogram (Figure 4) and statistics show that, based on simple TF-IDF cosine similarity, the content overlap between citing and cited papers is generally quite low (median ¡ 0.1). This suggests that relying solely on TF-IDF might miss many relevant citations and highlights the potential benefit of incorporating graph structure or more advanced semantic similarity methods.

5 Code Structure

The main scripts developed for this project are:

- `citation_graph_generator.py`: Builds the graph and performs Task 1 analysis.
- `precompute_data.py`: Generates offline TF-IDF model, vectors, and ID lists.
- `in_degree.py`: Calculates and analyzes node in-degrees from the graph file.
- `evaluation.py`: Implements the Task 2 prediction logic for a single input paper.
- `autograder.py`: Simulates the evaluation process using Average Recall@K.
- `analysis/similarity_score.py`: Calculates content similarity between connected papers.