CS768: Learning With Graphs Assignment

Yash Salunkhe - 210020156
Ravi Kumar - 210010052

GitHub Link: https://github.com/ravioli1369/cs768-assignment

1 Task 1

Approach for Data Extraction and Normalization

- **Paper Titles:** For each paper folder, the title was read from `title.txt`, normalized to lowercase, removed non-alphanumeric characters and extra whitespace.
- **Citations:** Bibliography entries were parsed from `.bbl` and `.bib` files using regular expressions to extract cited titles, which were normalized using the same process as above.
- **Citation Mapping:** Only citations matching titles within the dataset were considered valid edges.

Graph Construction

- The citation network was modeled as a directed graph using NetworkX.
- **Nodes:** Each paper (by normalized title).
- **Edges:** A directed edge from a paper to each cited paper present in the dataset.

Results

Number of Edges

The final citation graph contains **21,796 edges**.

Number of Isolated Nodes

Isolated nodes (papers neither citing nor cited) total **1,241**.

Degree Statistics

- **Average in-degree:** 3.33
- **Average out-degree:** 3.33

Degree Distribution

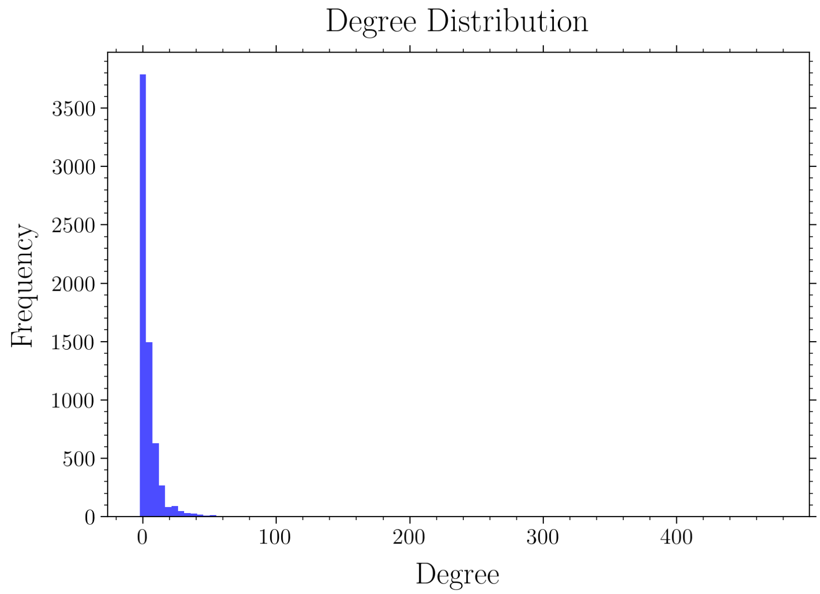

Figure 1: Histogram of node degrees (100 bins). Most nodes have low degree, with a long tail for highly cited/citing papers.

Diameter of the Graph

- The diameter was computed for each weakly connected component (ignoring edge direction).
- The **maximum diameter** among all components is **14**.

2 Task 2

Model Architecture

The core architecture implements a 2-layer GraphSAGE [1]model with the following components:

```
class GraphSAGELinkPredictor(nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super().__init__()
        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, out_channels)
        self.link_predictor = nn.Sequential(
            nn.Linear(out_channels*2, hidden_channels),
            nn.ReLU(),
            nn.Linear(hidden_channels, 1)
```

```
        )

    def encode(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p=0.2, training=self.training)
        x = self.conv2(x, edge_index)
        return x
```

Link Prediction Head

The decoder computes edge probabilities using concatenated node embeddings:

$$\phi(u, v) = \mathrm{MLP}(\mathrm{CONCAT}(z_u, z_v)) \tag{0.1}$$

where $z_u, z_v \in \mathbb{R}^{64}$ are node embeddings from the final GraphSAGE layer.

Training Configuration

Loss Function and Optimization

The model uses balanced binary cross-entropy loss:

$$\mathcal{L} = -\frac{1}{|E^+| + |E^-|}\left(\sum_{(u,v)\in E^+} \log \phi(u, v) + \sum_{(u',v')\in E^-} \log(1 - \phi(u', v'))\right) \tag{0.2}$$

Hyperparameters:

- Learning rate: 0.01 (Adam optimizer)
- Batch size: 2048 edges
- Dropout rate: 0.2
- Hidden dimension: 128
- Output dimension: 64

Text Processing Pipeline

```
def extract_features(paper_folders):
    vectorizer = TfidfVectorizer(max_features=300)
    texts = [preprocess(folder) for folder in paper_folders]
    features = vectorizer.fit_transform(texts).toarray()
    return features, paper_ids
```

Preprocessing steps include:

- Lowercasing and special character removal
- TF-IDF vectorization (300 dimensions)
- Text normalization using regex patterns

Evaluation Protocol

Recall@K Metric

$$\text{Recall@}K = \frac{|\text{Top-K Predictions} \cap \text{Actual Citations}|}{|\text{Actual Citations}|} \tag{0.3}$$

Inference Pipeline

New Paper Handling

1. Feature extraction using trained TF-IDF vectorizer
2. Temporary graph expansion with zero-initialized edges
3. Two-stage prediction:
    - Stage 1: TF-IDF similarity filtering (cosine > 0.25)
    - Stage 2: GNN scoring of top 200 candidates

```
def predict_citations(model, data, new_features, k=10):
    z = model.encode(data.x, data.edge_index)
    new_embedding = model.process_new_node(new_features)
    scores = model.link_predictor(
        torch.cat([new_embedding.expand(z.shape[0], -1), z], dim=1))
    return scores.topk(k).indices
```

Results

The recall at top-10 relevant citations for our model is ~**0.17**

References

[1] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.