Programming Assignment 3

Link Prediction and Retrieval on Graphs and Zero Shot Retrieval

Atishay Jain (210050026), Chaitanya Garg (210050039)

26th November 2024

1 Part - 1

1.1 Dataset

CoraFull and DeezerEurope datasets are used. Datasets set are **edge split** into train, test and validation sets in 60 : 20 : 20 ratio. Negatives are sampled within subgraphs.

1.2 GNN Training

Node pairs are classified into positive (those nodes which are connected by edges) and negative (not connected). For implementing, pytorch geometric was utilized for subgraph induction and embedding generation. The loss function was used based on cosine similarity, as defined in the equation (9) of this paper -

$$\text{loss}(\theta; \{\pi^w\}_{w\in V}) = \sum_{(u,v)\in E,(r,t)\in \bar{E}} ReLU\left(\Delta + s_\theta(r,t \mid \pi^r,\pi^t) - s_\theta(u,v \mid \pi^u,\pi^v)\right) \quad (9)$$

where Δ is a tunable margin, set to 1.

1.3 Non - Trainable Inference

Adamic Adar and Common Neighbors scores are non trainable and inferred directly from subgraphs.

1.4 GNN Inference

Node embeddings from the trained GNN models are used for predicting link on the test graph. A subset Q of 1000 nodes is selected such that each is part of atleast one triangle. For ranking the candidate node pairs, cosine similarity is used as the scoring function for each node in Q.

1.5 Random LSH-Based Inference

GNN Model outputs (node embeddings) are treated as Node features and used as input to LSH models. Random Hyperplanes are sampled and used to hash nodes based on their embeddings. Similarity of nodes in set Q are then only calculated against nodes with same hash.

1.6 Neural LSH-Based Inference

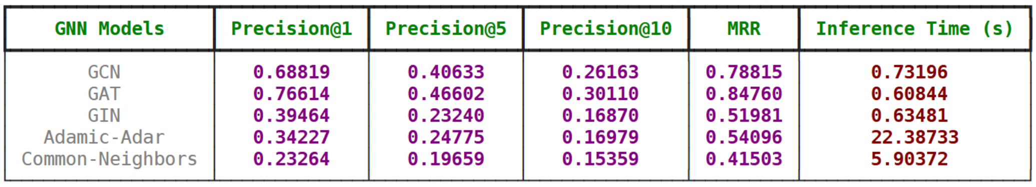
GNN Model outputs (node embeddings) are treated as Node features and used as input to LSH models.
A set number of hash tables are created. Hashes of node embeddings is caluclated in each table. Similarity of nodes in set Q are then only calculated against nodes with same hash in atleast 1 hash table.

1.7 Metrics Used

- **Precision@K (K = 1, 5, 10):** Measures the proportion of true positive edges in the top-K predictions.
- **MRR:** Reflects the rank quality of true connections among all predictions.

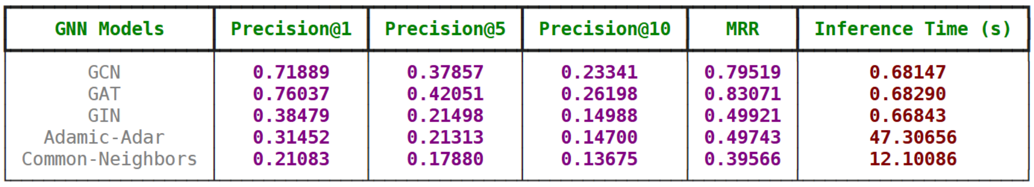
1.8 Results

1.8.1 GNN Inference



GNN Models	Precision@1	Precision@5	Precision@10	MRR	Inference Time (s)
GCN	0.68819	0.40633	0.26163	0.78815	0.73196
GAT	0.76614	0.46602	0.30110	0.84760	0.60844
GIN	0.39464	0.23240	0.16870	0.51981	0.63481
Adamic-Adar	0.34227	0.24775	0.16979	0.54096	22.38733
Common-Neighbors	0.23264	0.19659	0.15359	0.41503	5.90372

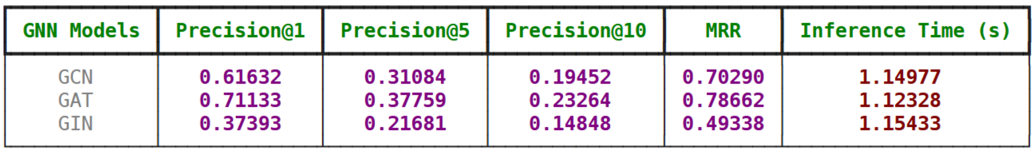
Figure 1: CoraFull dataset



GNN Models	Precision@1	Precision@5	Precision@10	MRR	Inference Time (s)
GCN	0.71889	0.37857	0.23341	0.79519	0.68147
GAT	0.76037	0.42051	0.26198	0.83071	0.68290
GIN	0.38479	0.21498	0.14988	0.49921	0.66843
Adamic-Adar	0.31452	0.21313	0.14700	0.49743	47.30656
Common-Neighbors	0.21083	0.17880	0.13675	0.39566	12.10086

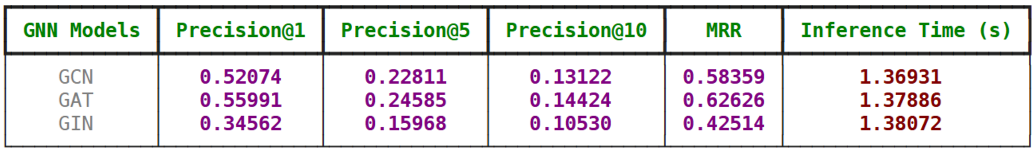
Figure 2: DeezerEurope dataset

1.8.2 Random LSH Inference



GNN Models	Precision@1	Precision@5	Precision@10	MRR	Inference Time (s)
GCN	0.61632	0.31084	0.19452	0.70290	1.14977
GAT	0.71133	0.37759	0.23264	0.78662	1.12328
GIN	0.37393	0.21681	0.14848	0.49338	1.15433

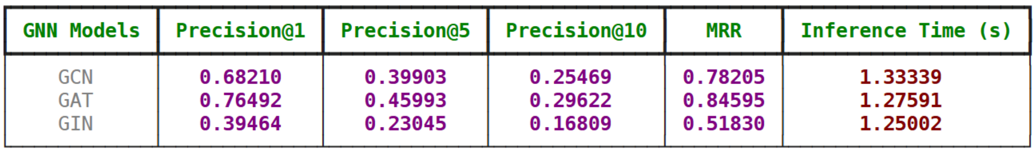
Figure 3: CoraFull dataset



GNN Models	Precision@1	Precision@5	Precision@10	MRR	Inference Time (s)
GCN	0.52074	0.22811	0.13122	0.58359	1.36931
GAT	0.55991	0.24585	0.14424	0.62626	1.37886
GIN	0.34562	0.15968	0.10530	0.42514	1.38072

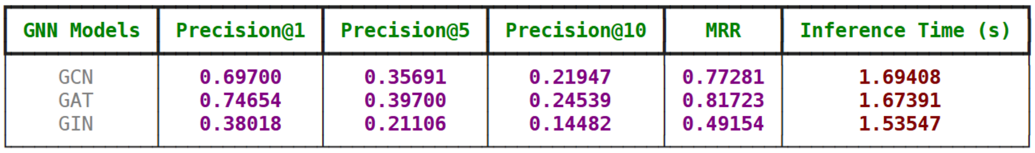
Figure 4: DeezerEurope dataset

1.8.3 Neural LSH Inference



GNN Models	Precision@1	Precision@5	Precision@10	MRR	Inference Time (s)
GCN	0.68210	0.39903	0.25469	0.78205	1.33339
GAT	0.76492	0.45993	0.29622	0.84595	1.27591
GIN	0.39464	0.23045	0.16809	0.51830	1.25002

Figure 5: CoraFull dataset



GNN Models	Precision@1	Precision@5	Precision@10	MRR	Inference Time (s)
GCN	0.69700	0.35691	0.21947	0.77281	1.69408
GAT	0.74654	0.39700	0.24539	0.81723	1.67391
GIN	0.38018	0.21106	0.14482	0.49154	1.53547

Figure 6: DeezerEurope dataset

1.9 Observations

- Non-trainable methods (Adamic-Adar and Common Neighbors) are significantly more computationally expensive compared to trainable models.
- LSH-based inference reduces the set of nodes against which cosine similarity is calculated. This reduction in the set of nodes is a reason for lower precision of Random LSH-based inference compared to direct GNN inference.
- Even with this reduction in set of nodes for similarity calculation, precision of LSH-based inference does not suffer significantly because nodes with with cosine similarity are more likely to end up in same hash buckets. Neural LSH-based inference has precision very close to GNN inference as the hyperplanes are tuned to get similar nodes in same hash bucket.
- Despite reducing the set of nodes for similarity calculations, the inference time for LSH-based methods is higher than that of GNN inference. This discrepancy may be attributed to the relatively small size of the graph datasets. In such cases, calculating similarities against all nodes is not prohibitively expensive, whereas the overhead of computing hashes and determining subsets of nodes in LSH-based methods dominates the inference time.
- Among LSH-based methods, Neural LSH is more computationally expensive than Random LSH. This is because Neural LSH requires locating nodes with the same hash across multiple buckets, introducing additional overhead.

2 Part - 2

2.1 Dataset

- **Hotpot-qa** dataset (from Assignment-1) was used to train the feed forward layer.
- **Scidocs** dataset was used for inference. It was processed to get Query-Document pair as positive or negative using the **Scidocs-qrels** dataset.

2.2 Model Architecture

The model architecture is designed to process input through a series of layers to produce a single scalar output. The steps are as follows:

1. **BERT for Masked Language Modeling (BERT-MLM):** The input is first passed through a pre-trained BERT model (`BertForMaskedLM`). The output corresponding to the `[CLS]` token from the final hidden layer is extracted as the feature representation.
2. **Fully Connected Linear Layer:** The `[CLS]` output is passed through a fully connected linear layer to transform it into a dense feature representation.
3. **ReLU Activation Function:** A Rectified Linear Unit (ReLU) activation function is applied to introduce non-linearity into the model.

4. **Dropout Layer:** To prevent overfitting, a dropout layer is applied, randomly setting a fraction of the layer's outputs to zero during training.
5. **Final Linear Layer:** The resulting features are passed through a final linear layer, which produces a single scalar output representing the model's prediction.

2.3 Training

- **Layer Freezing:** During training, all layers of the `BertForMaskedLM` model are frozen, as only the feed-forward layers are required to be trained.
- **Dataset:** Training is conducted using 1% of the `HotpotQA` dataset.
- **Negative Sampling:** For each positive example, 7 negative samples are generated.
- **Loss Function:** The training process employs the `CrossEntropyLoss` function to compute the loss.
- **Optimizer:** The `AdamW` optimizer is used to update the model parameters during training.

2.4 BaseCase Inference

- **Query Processing:** During inference, for each query, all positive and negative documents associated with the query are passed through the model simultaneously.
- **Ranking Generation:** The model computes scores for each document. These scores are used to rank the documents based on their relevance to the query.

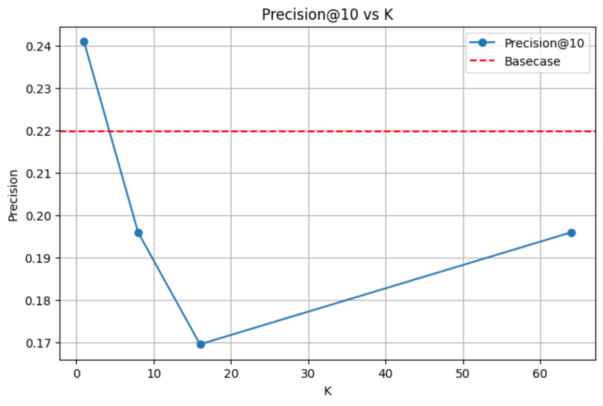
2.5 Modified Inference

- **Document-wise Processing:** For each query, each associated document is processed one at a time.
- **Parameter Management:** The original model parameters are copied to a local variable at the start of the inference process.
- **Masked Reconstruction:** For each query-document pair:
  - All model parameters are unfrozen.
  - A single step of masked reconstruction is performed on the `BERTForMaskedLM` model using the query-document input.
  - Inference is then performed using the reconstructed model, following the same procedure as described in the base case.
- **Parameter Restoration:** After processing every K query-document pairs, the model parameters are reset to their original values using the saved copy.

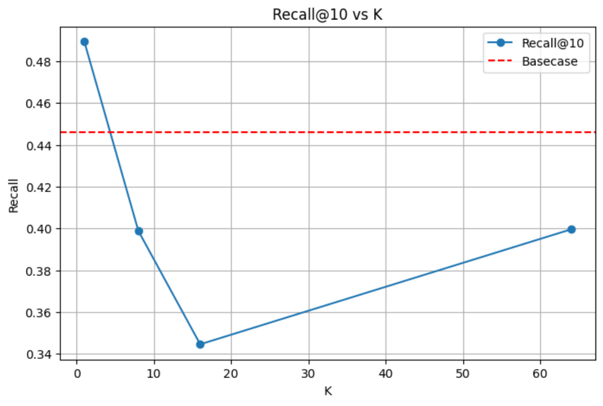
2.6 Result

Evaluation Scores	**Normal Inference**	**Modified Inference**			
		K = 1	**K = 8**	**K = 16**	**K = 64**
Precision@10	0.21979	0.24090	0.19589	0.16949	0.19589
Recall@10	0.44614	0.48943	0.39878	0.34453	0.39954
MRR	0.49045	0.51490	0.44592	0.27674	0.42016

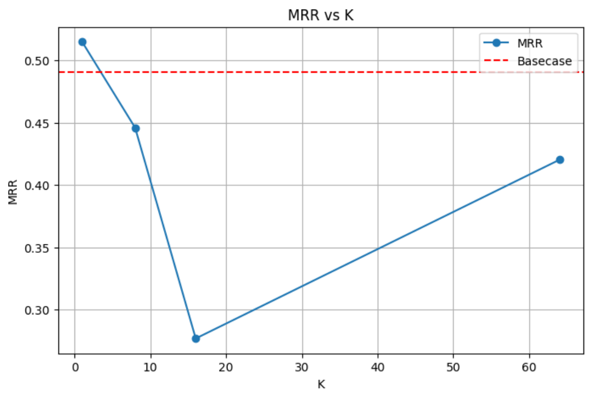
Table 1: Evaluation Scores



(a) Precision@10



(b) Recall@10



(c) MRR

Figure 7: Evaluation Scores

2.7 Observations

- The modified inference method performs best when $K = 1$. This observation suggests that retaining the reconstructed model from previous examples adversely affects its ability to make accurate inferences on the current input.
- This could be due to the fact that context from previous documents, whether positive or negative, may introduce a bias that is unrelated to the document currently being processed.
- **Reconstruction Input:** Using only the document (`doc`) for the masked reconstruction step may be more effective than using the query-document pair (`query+doc`). Including the query could bias the model by artificially strengthening query-document correlations, even for unrelated pairs. In contrast, using only the document helps the model build an unbiased contextual representation, improving its ability to accurately infer relevance during query-document inference.