textGNN-R: Graph Structural Text Retrieval

JUSTIN CARPENTER, Boise State university, USA

The problem with traditional approaches lie with the structural importance of a query. Traditional approaches do not consider the structure of a query nor document, only individual words or phrases. Recent works have explored the applicability of graphs to solve this problem. Graphs do not treat text as a sequence or sequences but as a set of co-occurrent words. [8] Previous works that utilize a graph corpus do not focus on ranking retrieved documents but on text classification. [9] In our proposed work we will be focused on using a graph for individual documents in our corpus and a graph for our queries. Then our Graph Neural Network (GNN) will be able to identify and rank our closest structured documents to our query structure. For our approach the first task will be converting a corpus, yahoo question and answer dataset, into individual graphs for each question. We will treat the answers as queries and also convert to a graphical representation. Using a basic graph convolution neural network from torch we will be able to train our network on the training split of our dataset. To test this we will use our testing split of new questions and identify the similarity of the answers. Using the reverse order to detect structure within an answer we propose a better structural choice than traditional approaches. To evaluate this proposition we will also implement an inverted index using terrier.

CCS Concepts: • **Information systems** → *Retrieval efficiency*; **Question answering**; **Novelty in information retrieval**; • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: Alternative Information Retrieval, neural networks

ACM Reference Format:
Justin Carpenter. 2022. textGNN-R: Graph Structural Text Retrieval. In *CS 637: Advanced Information Retrieval, Project 2, Spring 2022.* ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

Traditionally, information retrieval (IR) systems focus on document retrieval using keywords in a query. Typically the documents returned are ranked in a way using term frequency-inverse document frequency (TF-IDF), from the highest scoring document to the least.[7] This idea has been extended to incorporate specific phrases to search in order to obtain more specific results.

Inverted indexes are very large and require a large amount of preprocessing to create and must be re-run every time a new document is added to the corpus. Our proposed theory will be able to run on the entire corpus quickly and any new documents can be simply converted to a graph and added to the corpus without the need to re-run the other documents. We will attempt to answer the following three Research questions.

RQ1: Is a graph representation of a corpus smaller in size to store than a traditional indexing approach.

RQ2: Is a graph neural network better at retrieval of similar structures than BM25 and TF-IDF indexing.

RQ3: Is the time to rank the retrieved corpus an improvement over past approaches.

For our approach the first task will be converting a corpus, yahoo question and answer dataset, into individual graphs for each question. We will treat the answers as queries and also convert to a graphical representation. Using SubGNN [1], a Graph Neural Network(GNN) focused on classifying subgraphs within a graph. This model is one of the best neural network approaches for this task as a the corpus graph we made is well suited for subgraph classification. We will be able to train our network on the training split of our dataset. To test this we will use our testing split of new questions, not to be confused with the answers, as the answers for the entire corpus will be used in the training portion for subgraph labels. We do not want to predict new, unseen, labels (answers) but focus on matching the test questions with the correct answers. Using the reverse order to detect structure within an answer we propose a better structural choice than traditional approaches. To evaluate this proposition we will also implement an inverted index using terrier and perform ranking using TF-IDF and BM25.

2 RELATED WORK

Previous attempts of identifying text through graph representational learning or neural networks have had different approaches. Recent works such as using the cosine measure in neural networks for document retrieval [6] and learning to rank models using neural networks [3] have shown that there is an area with little exploration but high potential in accomplishing this type of task. The possibility to identify documents from search using graphs and additionally individual graphs per document [10] has shown that using graphs is a strong competition for information retrieval with results on pare with state of the art approaches. While these works use graphs to represent document retrieval using text, they fail to utilize graphs as a way to conserve space and computational cost. The difference between typical index or inverted indexing approaches used in pyTerrier[4], and more conventional search algorithms lie with the indexing and storage of such large corpus's. There are great advantages of conventional indexing are among availability, easy to use, and effective. Many state of the art approaches have focused more on optimizing indexes and inverted indexes for use in large scale datasets. While these optimizations have proven to be very effective, there is also some trade offs. Some being higher search times and others being more computational power in order to decode the encoded, compressed, stored index[5]. Although as seen the better the indexing or inverted-index, we see an increasing size of information needed to store such index. The fastest document retrievals rely on storing the index within memory, this can prove impossible in certain cases and difficult in others. The increase in information contained within indexes puts pressure on providers to increase hardware and results in the increase in cost [2].

CS 637:Advanced Information Retrieval, Project 2, Spring 2022

3 DATA DESCRIPTION

To thoroughly test our proposed solution, we had to choose a dataset containing a known ground truth and large in size in order to fully train our neural network. The dataset selected in these experiments is the Yahoo Question and Answer dataset. This dataset is a well known dataset that is available for any research for use. Question and Answering datasets are a well suited dataset for this experiments due to the known ground truth. Other datasets would be more benifitial through user studies in order to evaluate the returned results but due to constraints was not used in this research but may be in future works. In Yahoo Q&A each question has an answer, using the answer we are able to label this as the ground truth then selecting a part of the dataset to remain separate from the testing set in order simulate our dataset correctly.

3.1 Yahoo Question & Answer

Yahoo's Dataset consists of over 1.4 million questions and answers. Of these 1.4 million, we narrow down the dataset to 888,643 questions and answers that have more than 10 words. There are a total of 617,233 unique words in these questions and answers. Of these words we filter out 27,654 following the criteria in 4.1 Text Processing. Using the processed data, we proceed to use the answers matched with the answer number in the dataset to create our training data. Then using the questions of these answers as our validation and test for our approach.

4 METHOD

4.1 Text Processing

The first step in our method lies with processing the text prior to the neural network implementation. We first applied our corpus to some pre-processing text filters such as removal of stopwords, using NLTK English stopwords. This step limits cross graph connections to everyday words that have limited information gain. The second step was stemming words. With stemming the words, we remove the end of the words to focus on the root words. This removes duplicate words in different tenses. We also forgoer some text analysis to remove any additional high used words that appear to be outliers, used in almost all the corpus (answers). Finally by filtering out all grammar we can have our text ready to pass create our edge-lists for graph conversion.

The creation of a graph representation for all the answers and questions forgo the same process. The first step is to create the edge-lists, our edge-list will consist of each word in an answer matched to the answer's number. We will have the number of answers (888,643) plus the number of words (27,654) for a total of nodes (916,297). We calculate the frequency of each word in the answer in order to maintain the importance of each word in the answers, as shown in Figure1. This frequency is then used as our edge weight in our edge-list.

We can see that in the fist 10 answers that there is a high number of words corresponding to multiple other answers. This is then extended to the entirity of the dataset with growing number of connections. The lowest degree of any given word node representation will be 50 and answer node representation of 10 due to our pre-processing factor. This was due to a limiting factor of space to train our neural network the processed words must be selected and a smaller portion of the total size of the dataset can be used. We use only questions and answers with more than 10 words to filter out the "I don't know" and yes/no answers. This limited the dataset down 60%. We then choose to further limit the words selected after the previous filtering steps to words that occur in more than 50 questions or answers and words that occur in less than 50,000 questions or answers. This was chosen due to the lack of structural information provided given the large increase in cost.

4.2 Neural Network

While our text is ready to use, we must format it in order to use majority of graphical neural network models. The typical format is an edge-list, which requires some more formatting. We first must count all the term frequencies in each document. This frequencies is the edge weight for the term to the document. Once we have all the frequencies we can create our edge-list. The edge-list format is as followed term, document, term frequency. This is in familiarity to tf (term frequency) used in other implementations of inverted indexes. Given Figure 2 We can see an overview how given the processed text we create graphs for each question/answer matching the words to the question/answer they are within. These subgraphs are then put together as a larger graph which is our corpus. The weights of each edge is the frequency of that word in that document (TF). The large graph is then used in subGNN[1] as each subgraph being the classification to the answer number. The authors of SubGNN [1] explain the process of subgraph neural network classification relies on three layers. The first being position from one anchor point to another, this is where the decision to create the graphs using the answer's number as a node to create these points. Then the GNN will locate the k-hop neighborhood to identify the subgraph neighborhood structure. The final layer is dependent on individual node's neighbors.

4.3 Index Retrieval

For the Baseline of our experiments we use two well known retrieval methods. The two chosen are TF-IDF and BM25, which both methods rely on an indexed representation of the corpus. To create this index we use the pyTerrier[4] library which does a very good compression of the index for storage in mind and efficient retrieval.

5 EXPERIMENTS

To accurately test our method we used pyTerrier to compare size and complexity, also measure the Mean Reciprocal Rank (MRR) for effectiveness scoring. Using Terrier we can create an inverted index, a more typical way to index our corpus. This size will be in consideration and the initial time to compute the inverted index is significantly faster than compared to the time of training the graphical structured version. This study is not focused on the initial time complexity but for future reference it is considered. As for the way to compute the MRR for the textGNN-R approach we must first convert the questions then try to predict the answer, using the top 10 scored documents these were the top "ranked" documents to apply MRR to.

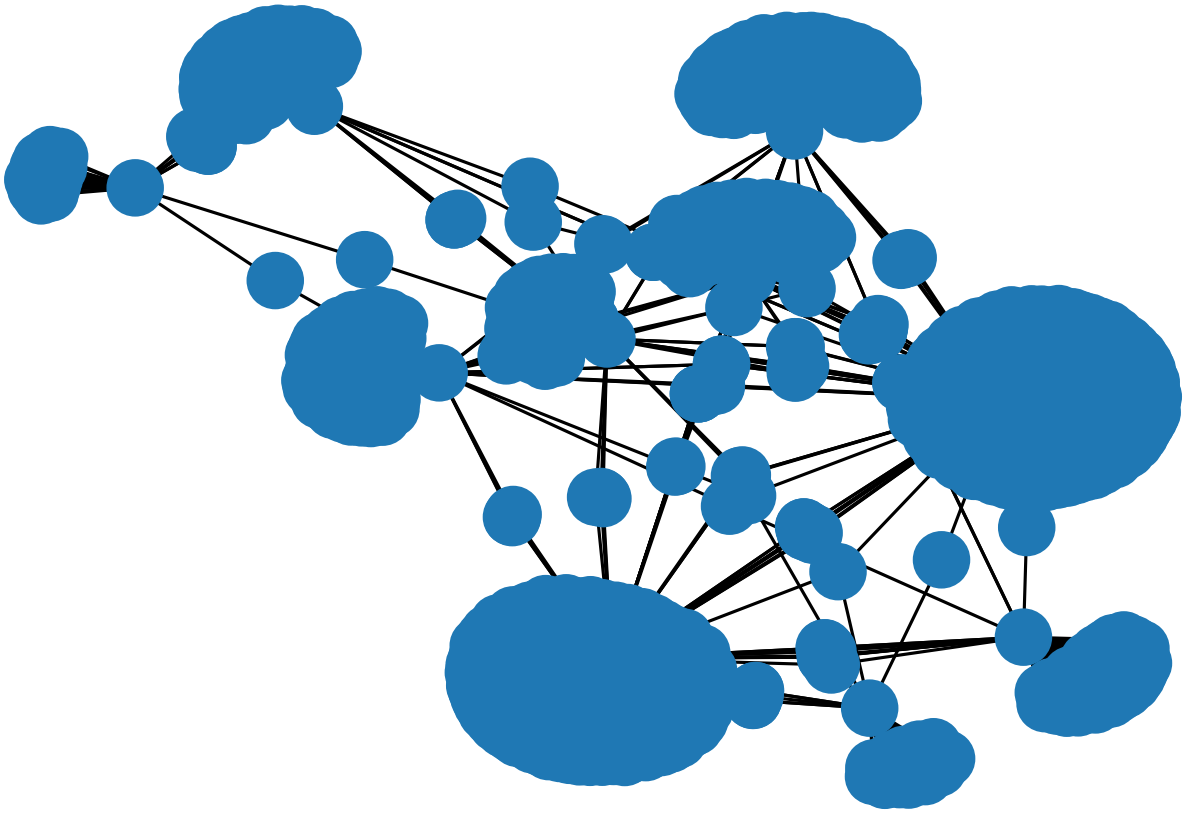

Fig. 1. All the words in the first 10 Answers shown as a graph representation

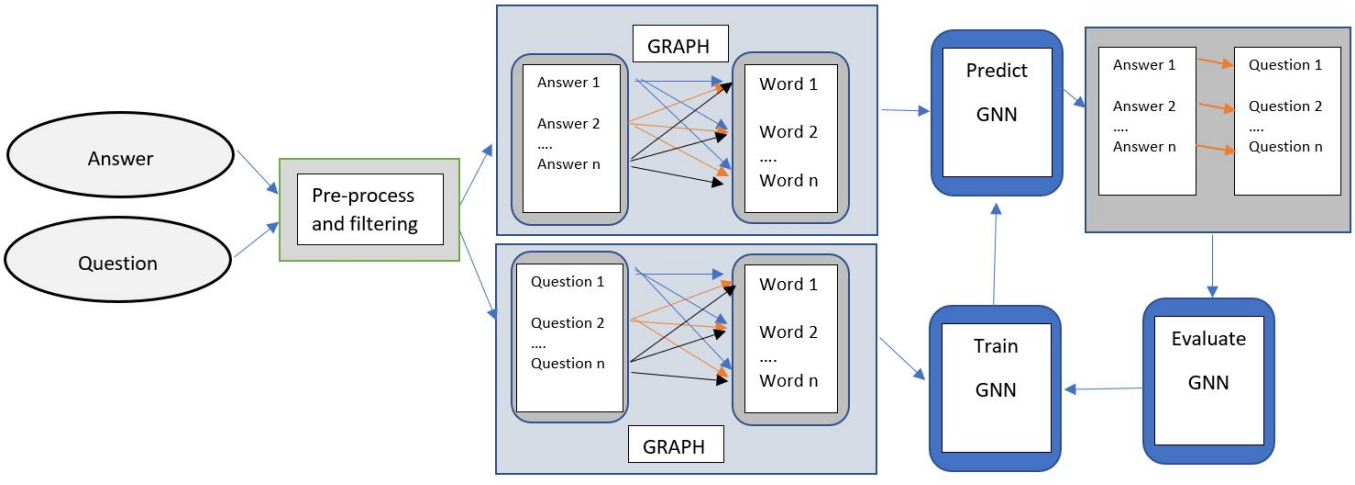

Fig. 2. High level design how the questions and answers are converted to graphs and used to train and predict what answers go to questions based on the word's structure.

5.1 Results

Table 1. Final Results measure's the Mean Reciprocal Rank (MRR), the size required to store the information needed for each method (Size) and the average retrieval time for each result (Retrieval-Time)

Evaluation			
Method	MRR	Size	Retrieval-Time
TF-IDF	0.287	244MB	13s
BM25	0.384	244MB	15s
SubGNN	0.193	29.5MB	18s

The results as shown in Table 1 were significant in two parts of this experiment. The MRR was significantly lower than BM25 by nearly 50% and over 30% compared to TF-IDF. The size required to store these methods were also a significant consideration with the index's requiring almost 90% more storage than the GNN. It is fair to say that both the GNN is stored with compression, typical storage of models, while the index's were also stored with compression. The size mentioned is the compressed size of both. The time to compute the ranking using these approaches is not significant enough to be of concern in our findings.

The poor MRR results of our experiment can be due to many factors. Some include the dataset selected. We chose this dataset due to the accessibility, time constraint, and funding. As a better dataset may be available using a larger collection paired with user studies. The user studies is necessary for majority of suitable datasets due to the requirement of labeled data needed for training a GNN. An additional reason why our findings were flawed was due to the computational power and memory size needed for training this model. We were limited to a single GPU to train this model and required weeks of ruining due to the size of the dataset. The training method had to be split into small batches which increased the runtime and could've impacted the final model. The final possible reason behind the lower results could lie with the choice in GNN models. We did not tune the hyperparamiters as efficiently as possible and this is not the only GNN model available. Future works can be among using a different Graph Structural model in order to better identify the document structures for paring.

6 CONCLUSION

During these experiments, we can conclude that using graphs to index is computationally equivalent to traditional methods of inverted indexes. As for the storage size, we can notice inverted indexes do require significantly more room to store even after compression. The final takeaway from the experiments is the MRR is lower compared to our approach, textGNN-R, compared to traditional inverted indexes using tf-idf and BM25. We conclude the loss of accuracy is more significant than the decrease in storage size. This is a novel approach for GNN in text retrieval which leads future work to improve the ranking score. We found that this is a plausible approach with room for significantly improvement.

REFERENCES

[1] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *Advances in Neural Information Processing Systems*, 33:8017–8029, 2020.

[2] J Shane Culpepper, Matthias Petri, and Falk Scholer. Efficient in-memory top-k document retrieval. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 225–234, 2012.

[3] Tom Kenter, Alexey Borisov, Christophe Van Gysel, Mostafa Dehghani, Maarten de Rijke, and Bhaskar Mitra. Neural networks for information retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1403–1406, 2017.

[4] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. Pyterrier: Declarative experimentation in python from bm25 to dense retrieval. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4526–4533, 2021.

[5] Willie Rogers, Gerald Candela, and Donna Harman. Space and time improvements for indexing in information retrieval. In *Proceedings of the Annual Symposium on Document Analysis and Information Retrieval*, 1995.

[6] Ross Wilkinson and Philip Hingston. Using the cosine measure in a neural network for document retrieval. In *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 202–210, 1991.

[7] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Trans. Inf. Syst.*, 26(3), jun 2008.

[8] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7370–7377, Jul. 2019.

[9] Yufeng Zhang, Xueli Yu, Zeyu Cui, Shu Wu, Zhongzhen Wen, and Liang Wang. Every document owns its structure: Inductive text classification via graph neural networks. *CoRR*, abs/2004.13826, 2020.

[10] Yufeng Zhang, Xueli Yu, Zeyu Cui, Shu Wu, Zhongzhen Wen, and Liang Wang. Every document owns its structure: Inductive text classification via graph neural networks. *arXiv preprint arXiv:2004.13826*, 2020.