Studying the Impact of Sparse Embeddings on Node Classification, Clustering and Link Prediction

Anirudh Penmatcha
Dept. of CS, Luddy SICE
Indiana University Bloomington
anpenma@iu.edu

N C Sathya
Dept. of CS, Luddy SICE
Indiana University Bloomington
satnc@iu.edu

Nischal B K
Dept. of CS, Luddy SICE
Indiana University Bloomington
nbangal@iu.edu

***Abstract*—Unsupervised graph embeddings like those created using algorithms like 'DeepWalk' and 'Node2Vec' have long been used to perform a plethora of machine learning tasks on graphs like Node Classification and Clustering. In this paper, we explore three techniques that allow us to sparsify these embeddings, these allow us to significantly reduce memory utilization with a negligible loss in performance across different tasks. The algorithm of choice for generating the initial embeddings is DeepWalk, selected for its relative simplicity. These initial embeddings serve as a baseline for comparison. Subsequently, we apply various sparsification techniques, including random elimination, thresholding, and top-K retention, to these embeddings. Our final embeddings occupy 67.9% less memory for a modest performance penalty of 1-2%. To validate our results, we perform extensive testing over three different tasks namely node classification, clustering and link prediction. Finally, we have averaged the final results across ten separate iterations to provide a holistic depiction of the effectiveness of our techniques.**

***Index Terms*—sparsification, graph embeddings, node classification, clustering, link prediction**

I. Introduction

The objective of this study is to investigate the impact of sparsity in embeddings on the accuracy of node classification, modularity score, and link prediction within a graph. This study aims to explore methods to maintain high accuracy levels while simultaneously reducing memory usage by employing sparse embeddings. In graph-based machine learning tasks, embedding methods play a crucial role in representing nodes as low-dimensional vectors. Traditional dense embeddings store information for all possible features, which can result in increased memory requirements, especially for large graphs. Sparse embeddings, on the other hand, leverage techniques to store only essential features, potentially offering a more memory-efficient solution without compromising performance. We will evaluate the effectiveness of sparse embeddings in node classification accuracy, modularity scoring, and link prediction performance. By examining these metrics, we aim to uncover insights into the trade-offs between memory efficiency and task performance when utilizing sparse embeddings in graph-based machine learning applications.

II. Previous Work

With respect to sparsifying the generated embeddings in a Graph most work so far revolves around GNNs. For example, [1] Peng et al. present a paper that investigates the application of state-of-the-art model compression techniques, including train and prune and sparse training, to reduce the computational and memory costs of Graph Neural Networks (GNNs). Another paper with similar research would be [2] Lutzeyer et al. where they explores the impact of sparsification on Message-Passing Neural Networks (MPNNs), proposing novel models, ExpanderGNN and Activation-Only GNN, to investigate sparse architectures in the Update step. Lastly, Chen et al. [3] investigate the performance of 12 graph sparsification algorithms across 16 graph metrics on 14 real-world graphs, revealing trade-offs between preserving different graph properties. Other than the previous one, we find that research on sparsifying Graphs in general has not been explored enough. Much has been done for GNNs, however.

III. Methodology

To generate node embeddings for our graph, we employ the DeepWalk algorithm, a popular method for learning representations of nodes in graphs based on random walks. The resulting embeddings provide dense representations of nodes in a low-dimensional space, capturing structural similarities between nodes based on their network connectivity patterns.

A. Generating Embeddings using DeepWalk

DeepWalk is a notable algorithm developed to generate node embeddings for graphs, turning graph structure into a format that machine learning algorithms can understand and use. Its fundamental operation mimics that of natural language processing, particularly the techniques used in Word2Vec, to learn representations of nodes based on their connectivity patterns. This method has proven particularly useful in the realm of social network analysis, where understanding and leveraging the intricate relationships between entities is important.

The key idea behind DeepWalk revolves around the use of random walks through the graph to capture the local neighborhood structures of each node. In practice, the algorithm begins by selecting a node at random and then simulates a walk by moving from one node to its adjacent node , according to a predefined probability distribution. These walks generate sequences of nodes similar to sentences in a text corpus, where the nodes play the role of words. We can adjust the length of these walks and the number of walks per node based on the size and complexity of the graph.

After creating these sequences, DeepWalk utilizes them as the training data for a neural network model that implements the skip-gram architecture from Word2Vec. The goal here is simple: predict the chance of encountering neighboring nodes. We provide the network with the correct sequence of nodes. This step is akin to predicting surrounding words given a specific word in a sentence. As the model trains, it learns to encapsulate the role and position of a node within the broader network context into a compact vector, or embedding. These embeddings can capture both the local and global position of nodes in the whole graph.

What makes DeepWalk particularly powerful is its ability to handle very large networks. It maps high-dimensional data (the graph) into a lower-dimensional space while preserving the structural nuances. This dimensionality reduction allows easier application of various machine learning tasks that may not handle high-dimensional data well, such as clustering, classification, or anomaly detection.

The computational efficiency of DeepWalk allows it to scale to networks with millions of nodes and edges, a typical scenario in today's data-rich environment. Adjusting parameters like walk length, number of walks, and the dimension of the embeddings can optimize its performance for specific tasks or datasets.

Yet, despite its strengths, DeepWalk is not without limitations. For instance, it assumes that the graph is static, which might not hold true for dynamic networks where edges and nodes can change over time. Moreover, it focuses on the structure of the network, overlooking node-specific attributes that could provide more insights.

B. Sparsification

A sparse embedding is analogous to a sparse matrix, where a majority of the elements in the embedding vector are zero. Sparsification is the process of converting a dense embedding into a sparse embedding by carefully removing redundant elements. This can be achieved using the three simple techniques detailed below:

- **Sparsification through Random Elimination:** This method involves randomly setting a fraction of embedding values to zero, effectively introducing sparsity by removing select connections between nodes in the embedding space.
- **Sparsification through Thresholding:** Here, we apply a threshold to the embeddings, setting values below a certain cutoff to zero while retaining higher magnitude values. This approach aims to retain significant features while discarding less important ones.
- **Sparsification through Top-K retention:** This technique involves retaining only the top-k largest values in each embedding vector, effectively prioritizing the most significant connections and discarding the rest.

1) Random Elimination: In this phase of our study, we investigate the effects of random sparsification on node embeddings by systematically removing a percentage of embedding values from each node. We vary the sparsity levels from 10% to 100%, incrementing by 10% at each step, to assess how different degrees of sparsity influence the performance of node classification, modularity scoring, and link prediction tasks. For each sparsity level (ranging from 10% to 100%), we randomly zero out the specified percentage of embedding values for every node in the graph.

2) Thresholding: In this phase of our research, we explore the impact of thresholding-based sparsification on node embeddings by retaining only values above specific limits. We evaluate the performance of node classification, modularity score, and link prediction tasks using different threshold values (0.05, 0.1, 0.25, 0.5, 0.75, and 1) to investigate how varying threshold levels influence the effectiveness of the sparsified embeddings. By applying thresholding, we selectively retain embedding values that exceed the defined threshold limit, thereby reducing the dimensionality and introducing sparsity into the embeddings. For each threshold value, we apply the thresholding technique to the node embeddings, retaining values above the specified threshold and setting others to zero.

3) Top-K Retention: In this phase of our investigation, we explore the impact of top-k sparsification on node embeddings by retaining only the top 'k' largest values in each embedding vector. This approach introduces sparsity by prioritizing the most significant embedding features based on their magnitude. We evaluate the performance of node classification, modularity score, and link prediction tasks using different values of 'k' (1, 5, 10, 25, 50, 75, 100, and 128) to understand how varying sparsity levels affect the utility and effectiveness of the sparsified embeddings. For each 'k' value, we select and retain only the top 'k' largest values in each node's embedding vector, setting the rest to zero.

IV. Experimental Setup

By exploring these sparsification methods, we seek to understand how the degree of sparsity affects the performance of downstream graph analytics tasks. For our experiments, we fix the following hyperparameters of the DeepWalk algorithm:

- Walk Length: Set to 20, determining the length of each random walk performed on the graph.
- Number of Walks: Configured to 80, representing the number of random walks initiated from each node to generate node sequences.
- Number of Embeddings: Chosen as 128, specifying the dimensionality of the learned node embeddings.

These fixed hyperparameters ensure consistency across our experiments, allowing us to isolate the impact of sparsity methods on the performance metrics of interest: node classification accuracy, modularity score, and link prediction efficacy. To better understand the effect of sparsification, we iteratively generate 3 sets of embeddings, each with 10 different embeddings. We apply each of our sparsification techniques on one of these sets. The results obtained from all the embeddings are then averaged out to help us better understand the average case performance of the new embeddings.

Through systematic evaluation and comparison, we aim to provide insights into the trade-offs between sparsity-induced

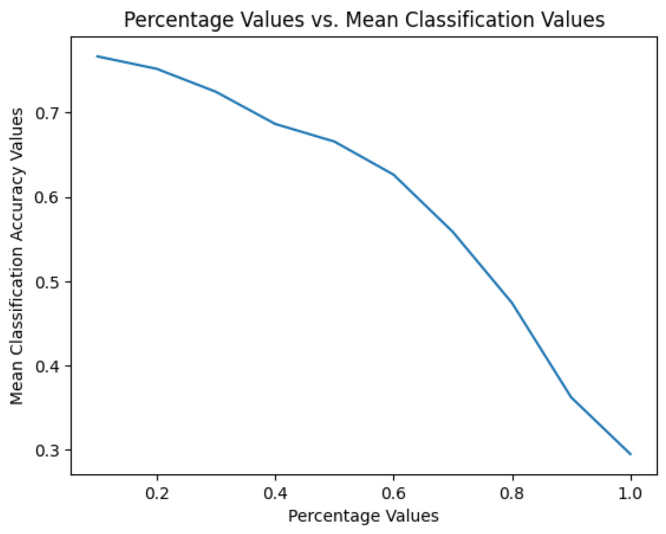

Fig. 1: Random Elimination Technique Mean Performance on Node Classification

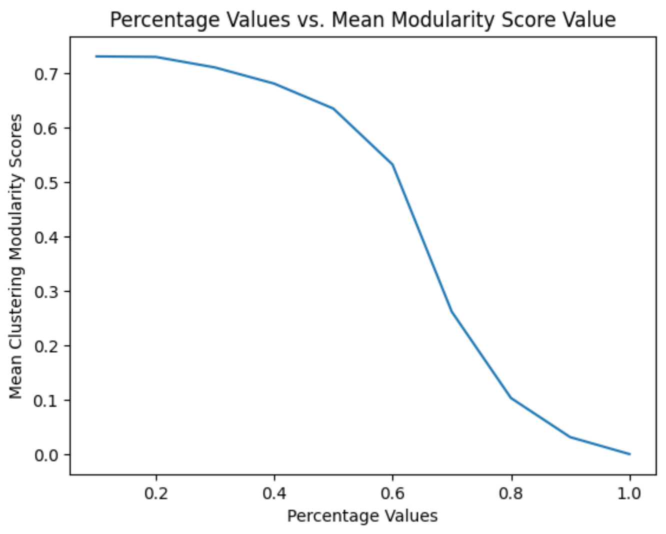

Fig. 2: Random Elimination Technique Mean Performance on K Means

memory savings and task-specific performance in graph-based machine learning applications.

V. Results

A. Random Elimination

In our comprehensive examination of random embeddings, we observed a persistent trend in classification accuracies, which consistently hovered around the 30% mark following the removal of varying proportions of embeddings. Concurrently, the modularity scores exhibited a stable performance, maintaining values within the range of 40-50% across different experimental conditions. However, the performance of link prediction was marked by notable instability, characterized by significant fluctuations and variability in the observed values throughout our rigorous analysis (Figure 1-6 show results of Random Elimination). Moreover, table 1 shows the difference in memory that by randomly removing 30% of the values we were able to reduce the memory by approximately 24% while maintaining reasonable accuracy.

TABLE I: Difference in Memory Observed due to Sparsification using Random Elimination

Length of Embedding	Bytes in Memory	Percentage
128	624	0
116	576	10
103	524	20
90	472	30
77	420	40
64	368	50
52	320	60
39	268	70
26	216	80
13	164	90
0	112	100

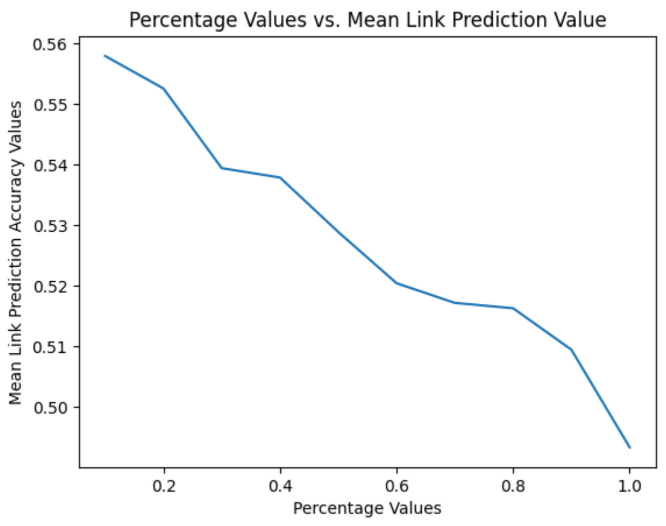

Fig. 3: Random Elimination Technique Mean Performance on Link Prediction

B. Thresholding

In our investigation involving thresholding, we observed that both modularity scores and classification accuracies remained relatively stable for threshold values of 0.05, 0.1, and 0.25, before exhibiting a noticeable change in trend. However, despite these observations, the performance of link prediction continued to exhibit significant instability throughout our analysis (Figure 7-12 show results of Thresholding). Moreover, table 2 shows the difference in memory that by considering absolute values greater than 0.25, we were able to reduce the memory by approximately 67.9% while maintaining reasonable accuracy.

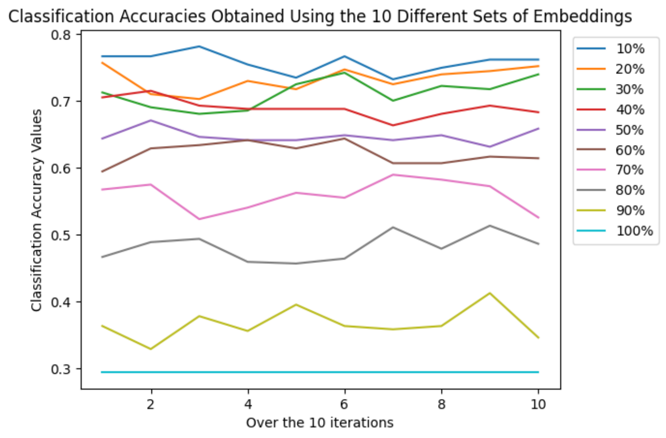

Fig. 4: Random Elimination Technique Performance for the 10 Embeddings on Node Classification

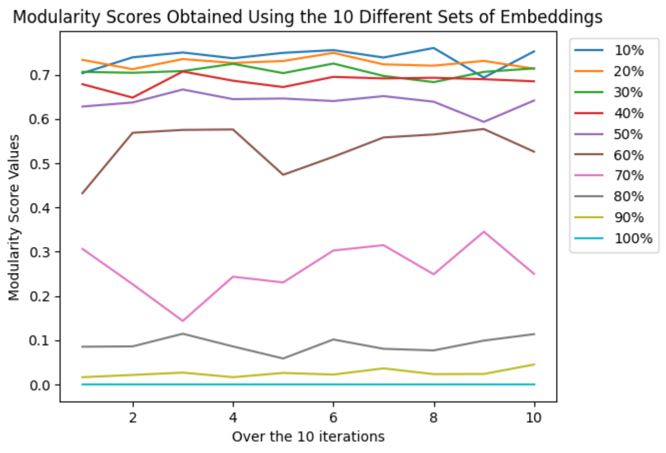

Fig. 5: Random Elimination Technique Performance for the 10 Embeddings on K Means

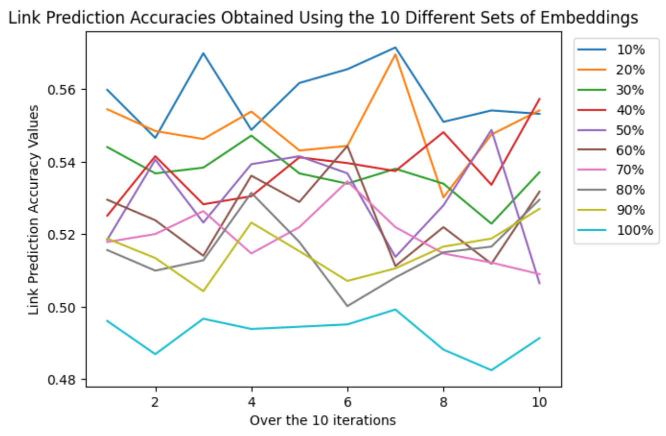

Fig. 6: Random Elimination Technique Performance for the 10 Embeddings on Link Prediction

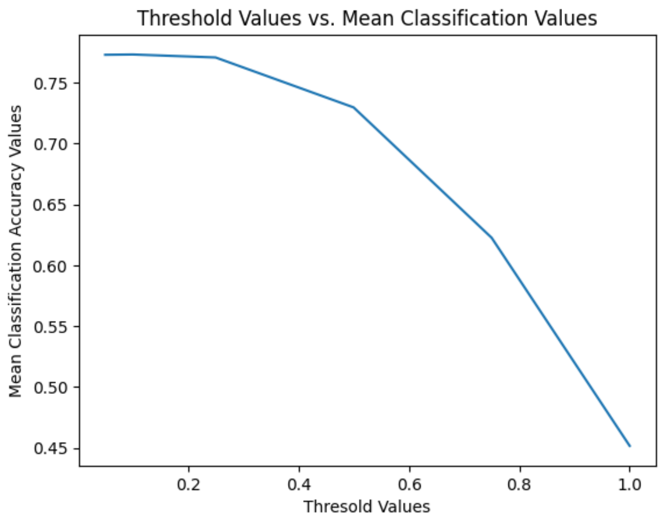

Fig. 7: Thresholding Technique Mean Performance on Node Classification

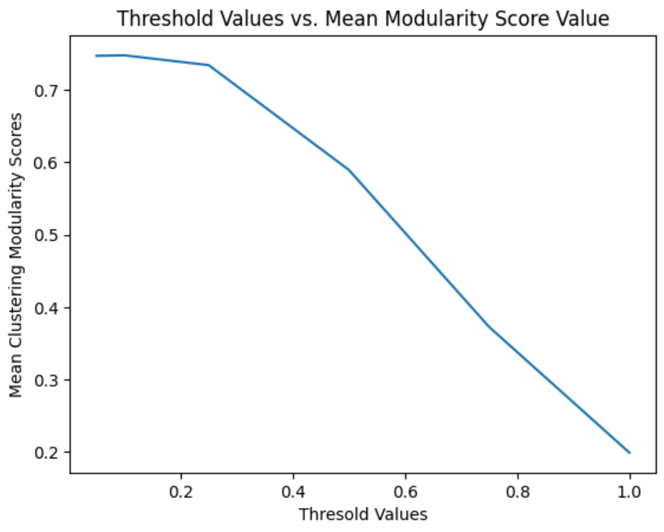

Fig. 8: Thresholding Technique Mean Performance on K Means

C. Top K Values

In the analysis of the top k values, we found that node classification accuracies and modularity scores maintained a consistent pattern up to the top 25 values before displaying a sharp decline. Meanwhile, the performance of link prediction remained inconsistent and exhibited ongoing variability across our experiments (Figure 13-18 show results of Top K). Moreover, table 3 shows the difference in memory that by considering top 25 values we were able to reduce the memory by approximately 66% while maintaining reasonable accuracy.

VI. Conclusions

We highlight the key finding that considerable reduction in the size of embeddings can be achieved without significant

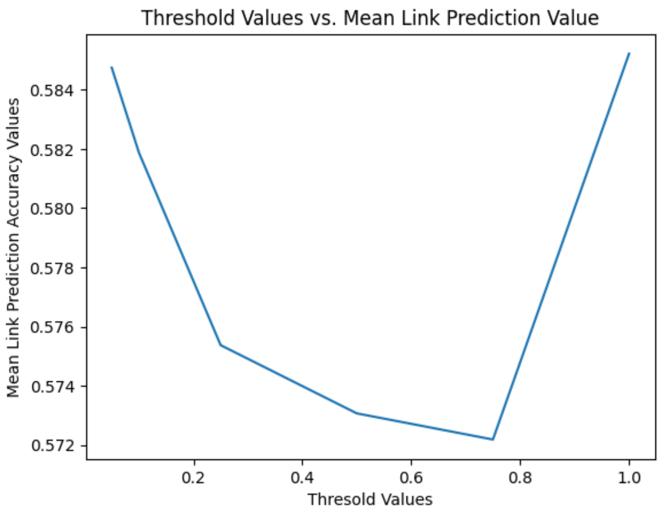

Fig. 9: Thresholding Technique Mean Performance on Link Prediction

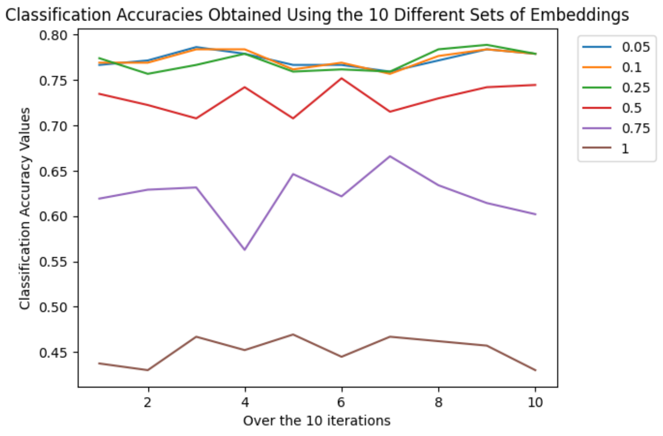

Fig. 10: Thresholding Technique Performance for the 10 Embeddings on Node Classification

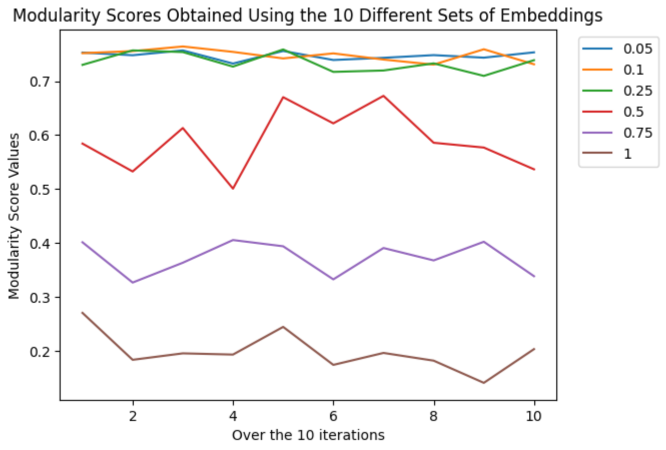

Fig. 11: Thresholding Technique Performance for the 10 Embeddings on K Means

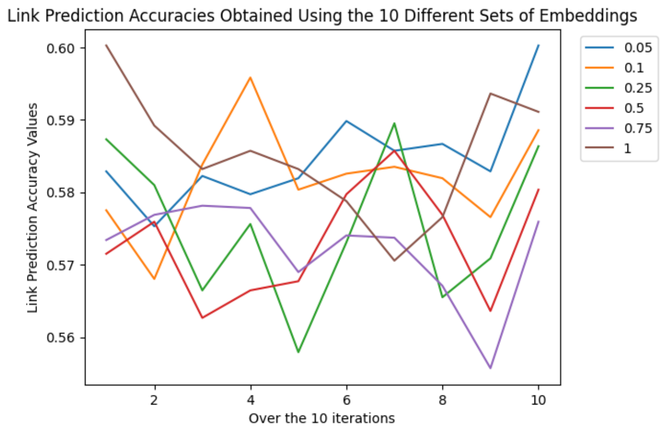

Fig. 12: Thresholding Technique Performance for the 10 Embeddings on Link Prediction

TABLE II: Difference in Memory Observed due to Sparsification using Thresholding

Length of Embedding	Bytes in Memory	Values
128	624	0
98	504	0.05
74	408	0.1
22	200	0.25
2	120	0.5
0	112	0.75
0	112	1

loss in accuracy. However, some of the interesting potential future work include scaling experiments to larger graph datasets to understand the scalability and generalizability of sparsified embeddings, and evaluating their accuracy across diverse graph sizes and complexities. Additionally, extending analysis to other popular graph embedding algorithms such as node2vec, Graph Convolutional Networks (GCN), and

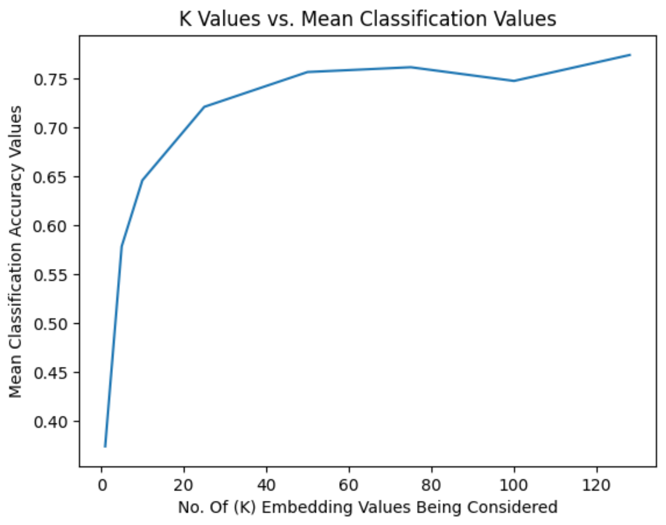

Fig. 13: Top K Technique Mean Performance on Node Classification

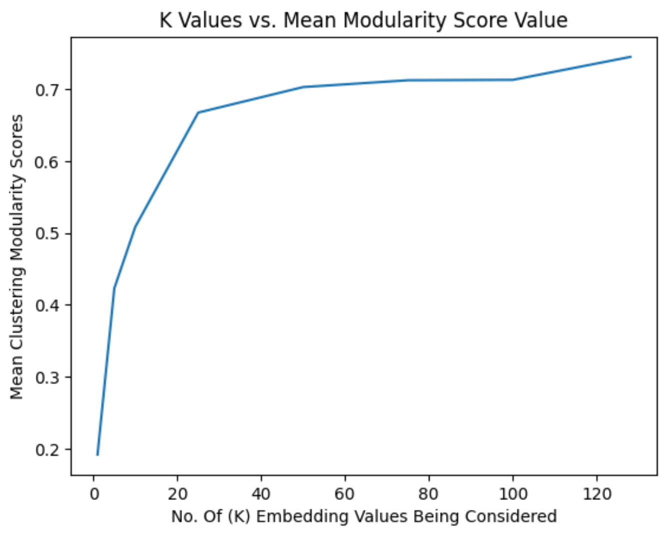

Fig. 14: Top K Technique Mean Performance on K Means

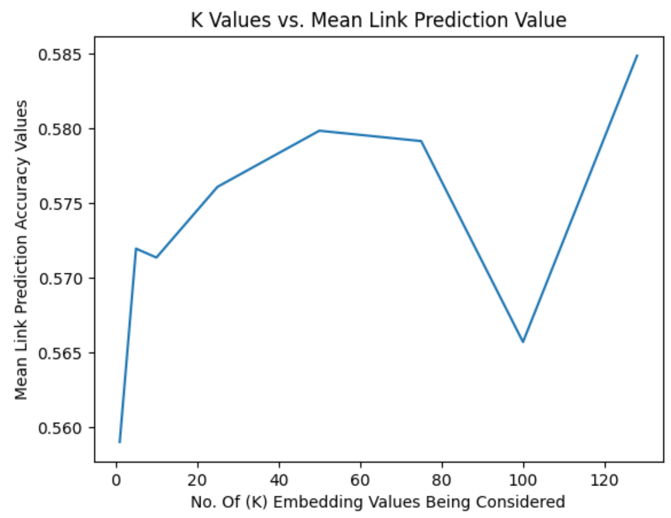

Fig. 15: Top K Technique Mean Performance on Link Prediction

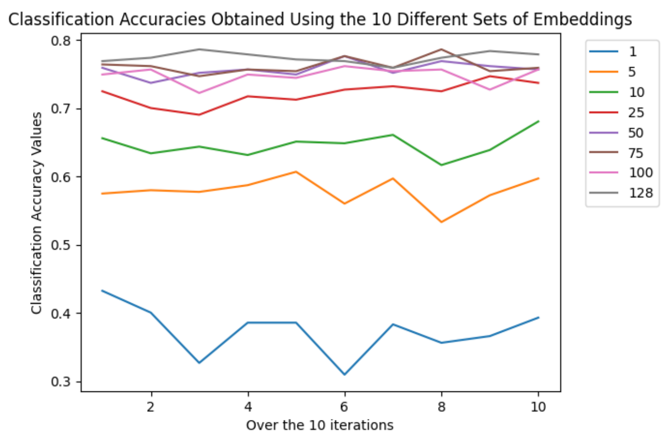

Fig. 16: Top K Technique Performance for the 10 Embeddings on Node Classification

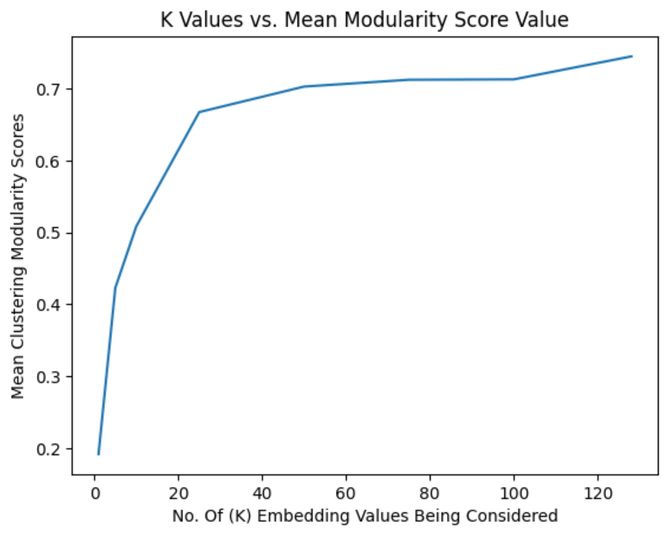

Fig. 17: Top K Technique Performance for the 10 Embeddings on K Means

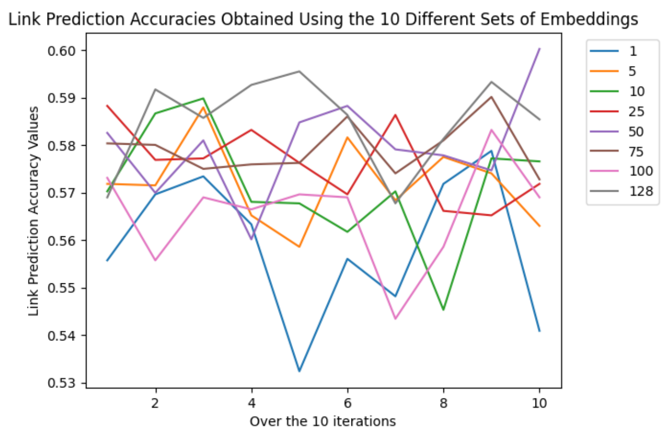

Fig. 18: Top K Technique Performance for the 10 Embeddings on Link Prediction

GraphSAGE will allow for a comprehensive comparison of sparsification techniques' impact on various graph representation methods. Investigating factors contributing to instability in link prediction tasks aims to refine sparsification methods and enhance reliability. Lastly, exploring the efficacy of sparsification on randomly generated graphs will elucidate its generalizability and applicability beyond real-world datasets.

VII. Team Member Contributions

Sathya: Conducted initial research for the project and wrote essential helper functions. Implemented baseline performance code, including random elimination and thresholding techniques. Additionally, contributed to creating select presentation slides.

Anirudh: Conducted initial research for suitable sparsification methods to work on. Enhanced existing helper functions for improved project suitability. Developed code to generate

TABLE III: Difference in Memory Observed due to Sparsification using Top K

Length of Embedding	**Bytes in Memory**	**Values**
128	624	128
100	512	100
75	412	75
50	312	50
25	212	25
10	152	10
5	132	5
1	116	1

10 distinct embeddings to test the code on. Implemented the Top K algorithm and organized and rewrote most of the sparsification methods for the final codebase to efficiently generate results and also wrote code for generating the various plots and tables.

Nischal: Evaluated multiple datasets to select the most appropriate for the project's scope. Explored various sparsification and embedding generation methods. Conducted a comprehensive review of prior research. Calculated memory utilization and restructured the codebase for clarity. Took a lead role in crafting the presentation and report.

References

[1] H. Peng, D. Gurevin, S. Huang, T. Geng, W. Jiang, O. Khan, and C. Ding, "Towards Sparsification of Graph Neural Networks," arXiv:2209.04766 [cs.LG], 2023.

[2] Johannes Lutzeyer, Changmin Wu, Michalis Vazirgiannis. "Sparsifying the Update Step in Graph Neural Networks." ICLR Workshop on Geometrical and Topological Representation Learning, Apr 2022, Online, France. ffhal-04447629f

[3] Yuhan Chen, Haojie Ye, Sanketh Vedula, Alex Bronstein, Ronald Dreslinski, Trevor Mudge, and Nishil Talati. 2023. Demystifying Graph Sparsification Algorithms in Graph Properties Preservation. Proc. VLDB Endow. 17, 3 (November 2023), 427–440. https://doi.org/10.14778/3632093.3632106

[4] Sarwan Ali, Muhammad Ahmad, Maham Anwer Beg, Imdad Ullah Khan, Safiullah Faizullah, and Muhammad Asad Khan. 2024. SsAG: Summarization and Sparsification of Attributed Graphs. ACM Trans. Knowl. Discov. Data 18, 6, Article 141 (July 2024), 22 pages. https://doi.org/10.1145/3651619

[5] Zirui Liu, Kaixiong Zhou, Zhimeng Jiang, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. 2023. "DSpar: An Embarrassingly Simple Strategy for Efficient GNN training and inference via Degree-based Sparsification." Transactions on Machine Learning Research. https://openreview.net/forum?id=SaVEXFuozg.