Data Mining Homework 4

Alessio Maiola

1 Node Classification on Cora

1.1 Data Preprocessing

For the first task, I imported the Cora dataset and preprocessed it. I extracted the largest connected component of the dataset and implemented a custom split of the nodes (60% of nodes are used for the train set, while the remaining 40% are evenly split between validation and test set). Finally, I normalized the features of the dataset.

1.2 Models

I implemented a modular model, which can be used with three kinds of Graph Convolutions.

The number of layers can be configured. Intermediate layers are composed by blocks constituted by:

- a graph convolutional layer, applying one of the supported convolution types;
- the SiLU activation function, which mitigates the risk of having dead neurons in deeper networks;
- a dropout layer

The last two layers of the network are a final embedding layer, with ReLU activation, and a last convolution to apply Softmax and match the output dimension. Hence, the minimum number of layers of my architecture is two.

The first class I used for graph convolutions in my model is GCNConv. It is a classic graph convolutional layer that operates by aggregating information from a node's neighbors and its own features. Without using edge weights, the node-wise formulation of the layer is the following:

$$h_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{d_i d_j}} W^{(k+1)} h_j^{(k)}\right)$$

where d_i, d_j are the degrees of the nodes i and j and σ is an activation function.

The second class I employed for graph convolutions is SAGEConv, which aggregates feature information with a slightly different technique, using different weights to combine node and neighbors' information.

$$h_i^{(k+1)} = \sigma\left(W_1^{(k+1)} h_i^{(k)} + W_2^{(k+1)} \sum_{j \in \mathcal{N}(i)} \frac{h_j^{(k)}}{|\mathcal{N}(i)|}\right)$$

The last class which can be selected to populate convolutional layers is GATConv. This class is characterized by the use of attention parameters α_{ij}, providing additional weights to neighbors contributions.

$$h_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{ij} W^{(k+1)} h_j^{(k)}\right)$$

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T \left[W^{(k+1)} h_i^{(k)} \,||\, W^{(k+1)} h_j^{(k)}\right]\right)\right)}{\sum_{l \in \mathcal{N}(i) \cup \{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T \left[W^{(k+1)} h_i^{(k)} \,||\, W^{(k+1)} h_l^{(k)}\right]\right)\right)}$$

Where $\mathbf{a}$ is a learnable attention vector parametrizing the attention mechanism and $||$ denotes the concatenation or the sum operation, depending on the chosen concat parameter.

It is important to notice that increasing the depth of the network aggregates information from increasingly distant nodes. While this allows the model to capture global patterns and long-range dependencies in the graph, it may also lead to the mixing of too much information in the features of individual nodes. This phenomenon may make it harder for the model to distinguish between the unique attributes of different nodes, as their feature representations become more and more similar. Given the small size of the Cora graph, whose number of nodes in the largest connected component is below 2500, networks with fewer layers achieve better performances.

1.3 Experiment setup

I fine-tuned the network's hyperparameters using grid search. As anticipated, shallower networks outperformed deeper ones, with 2-layer architectures achieving higher metrics compared to deeper models. Additionally, for all deeper networks, the optimal number of layers was found to be 3, further demonstrating how increased depth negatively affects performance.

Moreover, I also used class weights to tackle the imbalance of the original dataset. However, this had a minimal impact on performance. The formula used for the weights is the following:

$$w_i = \frac{N}{c \cdot n_i}$$

where N is the number of nodes, n_i is the number of samples of class i and c is the number of classes. Thanks to these weights, the impact on the loss of samples of any class i will be uniform and equal to $\frac{1}{c}$.

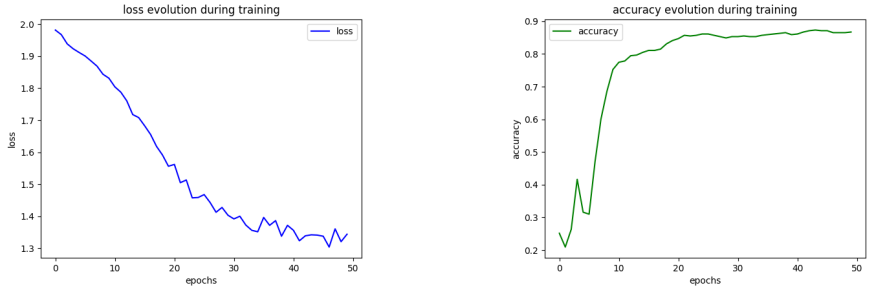
1.4 Results

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
GCN	0.8612	0.8718	0.8612	0.8628	0.9830
SAGE	**0.8672**	0.8766	**0.8672**	**0.8681**	**0.9854**
GAT	0.8652	**0.8786**	0.8652	0.8669	0.9845
DeepGCN	0.8612	0.8737	0.8612	0.8638	0.9801
DeepSAGE	0.8571	0.8630	0.8571	0.8579	0.9808
DeepGAT	0.8551	0.8631	0.8551	0.8564	0.9818
Node2Vec	0.8109	0.8188	0.8109	0.8112	0.9570

Table 1: Performance Metrics on Cora Dataset with weighted loss

From the results presented in the table, it is evident that the best-performing model is the 2-layered architecture utilizing SAGE convolutions. This network not only achieves the highest performance across several metrics, such as accuracy, recall, F1-score, and ROC-AUC, but also demonstrates remarkable computational efficiency.

When compared to GCN convolutions, the SAGE model achieves comparable efficiency, while being significantly more computationally efficient than GAT convolutions. This balance of performance and efficiency establishes the SAGE convolution-based architecture as the most suitable model for the node classification among all those tested.



(a) Loss. (b) Test Accuracy.

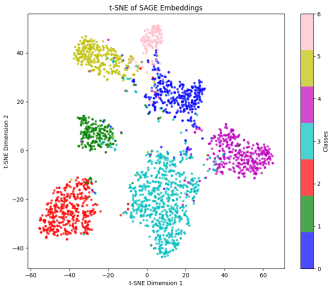
Figure 1: Metrics evolution over training.

1.5 Embedding and Node2Vec comparison

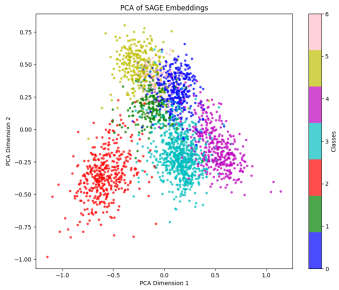
I implemented Node Classification using the Node2Vec implementation from the PyTorch Geometric library. By default, the model in PyTorch Geometric uses scikit-learn's Logistic Regression in the test function to assess the quality of the embeddings. However, to improve efficiency and avoid the overhead of repeatedly transferring tensors between GPU and CPU, I developed a simple

Logistic Regression model from scratch. This custom model is trained alongside the Node2Vec embeddings, enabling the entire process to remain on the GPU and eliminating unnecessary data transfers. Furthermore, I incorporated early stopping to interrupt the training process when the model's performance stopped improving.

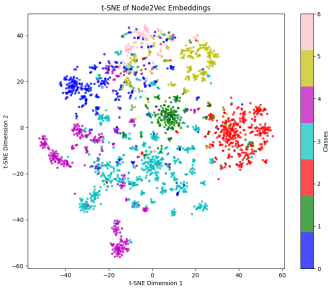
Despite these optimizations, the resulting Node2Vec model achieved worse performance in node classification, making it less suitable for the task compared to the other architectures. However, combining Node2Vec embeddings with a more complex classifier, rather than using Logistic Regression, could potentially improve the results. Nonetheless, this combined model is already the most computationally expensive option.



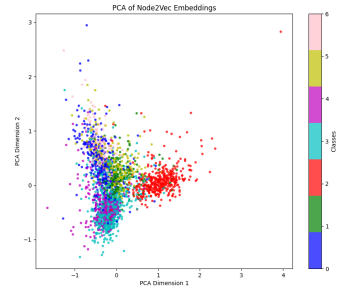
(a) Embeddings of SAGE (t-SNE)



(b) Embeddings of SAGE (PCA)



(c) Embeddings of Node2Vec (t-SNE)



(d) Embeddings of Node2Vec (PCA)

Figure 2: Node embeddings comparison

Each model in my architectures is designed to embed nodes in a way that explicitly emphasizes the separation of nodes belonging to different classes. This separation is achieved by leveraging node features and graph structure to enhance class distances in the embedding space. In contrast, Node2Vec takes a fundamentally different approach.

It aggregates nodes that are deeply connected or share similar random walk patterns, which often results in clusters of nodes that are topologically close,

but not necessarily distinct in terms of class labels. Additionally, Node2Vec does not utilize node features, which are critical for effectively distinguishing between classes. As a result, its embeddings are less appropriate for tasks like node classification, where the combination of graph structure and feature information is essential for optimal performance.

2 Generalization on other datasets

For this task, I performed a series of experiments, testing how my models behaved on other graph datasets.

The CiteSeer and PubMed datasets were imported using the same preprocessing as the Cora dataset. However, the models cannot be directly tested on these datasets, as they have different numbers of features and classes.

Instead of mapping the classes of the Cora dataset to the classes of the other datasets, which would require guessing the appropriate mapping criteria, I employed Singular Value Decomposition (SVD) to project the weights of the final layer into a reduced-dimensional space, reducing the dimensionality from $N \times d$ to $N \times r$, where d is the number of classes of the Cora dataset and r is the number of classes of the new dataset. This approach was feasible because CiteSeer and PubMed have fewer classes than Cora (6 and 3 versus 7, respectively).

Dealing with the input was more complicated, as CiteSeer has 3703 features compared to Cora's 1433, while PubMed has only 500 features.

Therefore, I tested two dimensionality reduction techniques on CiteSeer (SVD and Random Projection) and two dimensionality augmentation techniques (Random Projection, this time increasing the dimensionality of the dataset, and zero padding).

Finally, I also tried to freeze the hidden layers and replace the first and last layers of the architecture with new layers having the appropriate dimensions. Then, I retrained the new layers using the same hyperparameters identified for the models trained on the Cora dataset.

This technique required to fully retrain the 2-layer models from scratch, while the central layers could be kept unchanged for deeper architectures.

Overall, the models struggled significantly with generalizing from the Cora dataset to the CiteSeer and PubMed datasets. Despite attempting various solutions, none of the experiments achieved satisfactory results. The models consistently underperformed on the new datasets, with most metrics falling short of what could be considered acceptable generalization.

2.1 CiteSeer

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC	Method
GCN	**0.2858**	0.1977	**0.2858**	0.2032	0.5062	SVD
SAGE	0.1613	0.1540	0.1613	0.1098	**0.5727**	SVD
GAT	0.2382	**0.2683**	0.2382	**0.2194**	0.5659	Retrain
DeepGCN	0.2420	0.2362	0.2420	0.1851	0.5167	SVD
DeepSAGE	0.1802	0.1060	0.1802	0.0579	0.4726	SVD
DeepGAT	0.1712	0.1169	0.1712	0.1177	0.4637	Random Projection

Table 2: Performance Metrics on CiteSeer Dataset with weighted loss

From the above results, it is clear that the models do not perform well on the CiteSeer dataset. Even if some models, particularly those using SVD, perform slightly better than random guessing, their performance remains unsatisfactory and does not solve the task.

Despite both Cora and CiteSeer representing features as bag-of-words vectors and belonging to a similar domain of paper citations, significant differences in their characteristics make generalization between them challenging. While both datasets share a comparable number of classes, the class semantics differ considerably, with no direct mapping between the labels. Additionally, CiteSeer's feature representation spans over 3700 dimensions, more than twice that of Cora's 1433, necessitating substantial dimensionality reduction to enable compatibility.

Beyond features, differences in the structural properties of the graphs may hinder the transferability of learned representations. These combined factors underline the difficulty of achieving generalization from Cora to CiteSeer despite their apparent domain similarity.

Last, it is interesting to notice how even retraining the whole model with the new dataset usually performs poorly as well. This could probably be improved with hyperparameter tuning, but this emphasizes even more the difficulty of producing a general architecture that generalizes across different graphs.

2.2 PubMed

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC	Method
GCN	0.3791	0.3844	0.3791	0.3652	0.5229	Retrain
SAGE	0.3505	0.4259	0.3505	0.2707	0.6121	Random Projection
GAT	0.3956	0.3655	0.3956	0.2501	0.5492	Retrain
DeepGCN	**0.4574**	**0.4630**	**0.4574**	**0.4455**	0.6143	Random Projection
DeepSAGE	0.3822	0.3034	0.3822	0.3367	0.4758	Random Projection
DeepGAT	0.2287	0.2545	0.2287	0.1128	**0.6359**	Retrain

Table 3: Performance Metrics on PubMed Dataset with weighted loss

The differences between Cora and PubMed extend beyond feature representation and include substantial structural and semantic variations. While both

datasets share a general domain of citation networks, their feature representations differ in meaning and scope. Cora uses bag-of-words vectors where features represent word occurrences in paper abstracts. In contrast, PubMed's features are derived from TF-IDF scores of medical terms.

The structural properties of the graphs also diverge significantly. PubMed contains a much larger graph with many more nodes and edges compared to Cora, reflecting a more complex citation network structure. This disparity further complicates the transfer of learned representations across datasets.

Another notable difference is the number of classes: PubMed has only three classes, compared to the seven in Cora. This distant class structure, combined with the semantic mismatch of features and the structural differences in the citation networks, presents significant challenges to achieving generalization between the two datasets.

However, in contrast to CiteSeer, some deep architectures seem to perform slightly better than random and achieve noticeable accuracy values (e.g., 45%, compared to 33% for uniformly distributed random guessing). This, however, is still far from sufficient to claim that the model is close to solving the task.

3 Explainability

For this task, I primarily leveraged the code provided during the lab sessions to train the GNN Explainer model and visualize the results. The visualization highlights the weights of the links connected to the node under analysis, offering insights into the model's interpretability.

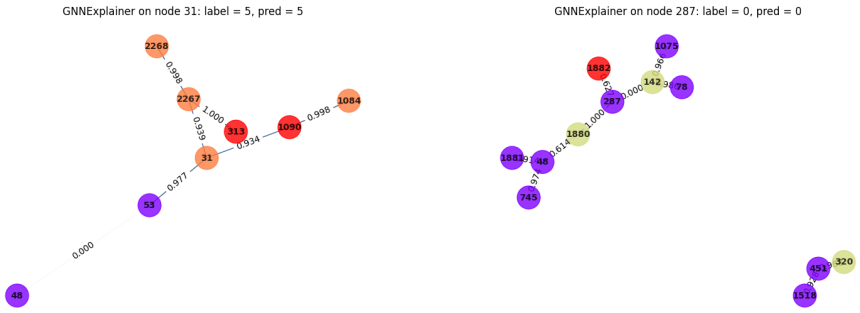
For the analysis, I used the SAGE model, which achieved the best performance for node classification. I analyzed at most 10 nodes from each of the following groups using the explainer:

- **Uncertain Correct**: Nodes correctly classified by the model but with high uncertainty (maximum probability below a threshold of 0.25).
- **Certain Wrong**: Nodes misclassified by the model with high confidence (maximum probability above a threshold of 0.35).
- **High Degree**: Nodes with the highest degrees in the graph (specifically, degrees exceeding the maximum degree minus 135).
- **Low Degree**: Nodes with a degree equal to 1.
- **Influential**: The top 10 nodes sorted by betweenness centrality.

The selected nodes were analyzed to understand the contributions of their neighboring connections and features, as revealed by the explainer.

3.1 Uncertain Correct

What emerged from this analysis is that nodes where the model struggles the most in classifying them are those with strong links to nodes from other classes. This is clear in the following examples:

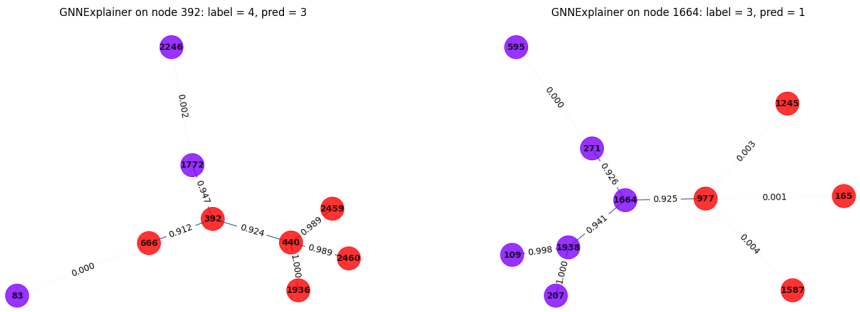


(a) Node 31 (b) Node 287

Figure 3: Explainer results for uncertain correct nodes.

3.2 Certain Wrong

In this analysis, as well as the previous one, mistakes in the prediction are usually caused by strong contributions from nodes of different classes, which lead the model to misclassify some of the nodes.



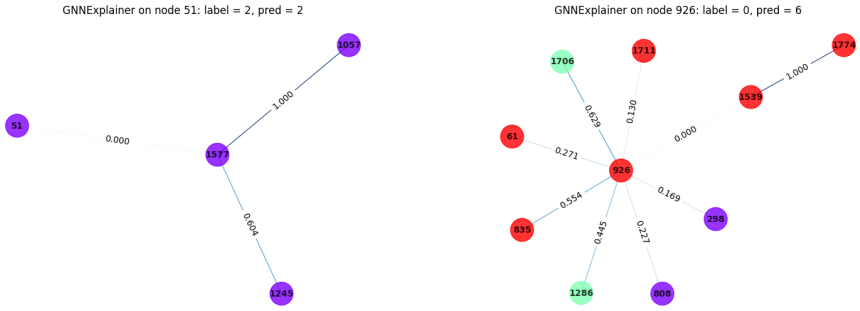
(a) Node 392 (b) Node 1664

Figure 4: Explainer results for certain wrong nodes.

3.3 Low/High Degree

When the model has very limited information from neighboring nodes (i.e. a single edge), it often performs remarkably well. In such cases, the model tends to assign a weight close to 0 to the edge, effectively classifying the node based

solely on its features. Conversely, for high-degree nodes, the model may aggregate excessive information from their neighbors, which can sometimes lead to misclassification.

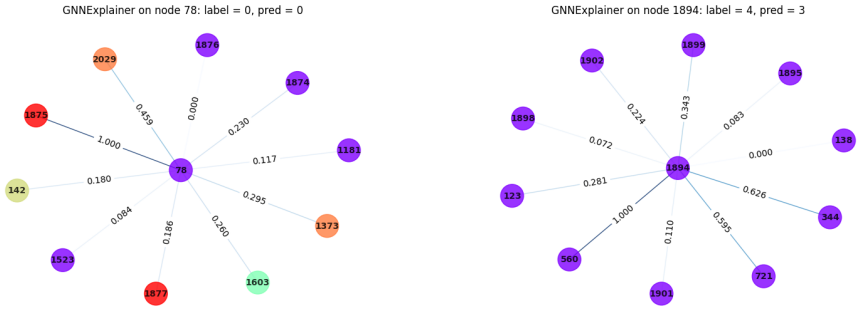


(a) Low degree node (Node 51) (b) High degree node (Node 926)

Figure 5: Explainer results for low/high degree nodes.

3.4 Top 10 nodes by betweenness centrality

These nodes significantly overlap with the high degree nodes and actually have the same mistakes. Anyway, most central nodes are usually classified correctly.



(a) Correctly classified node (Node 78) (b) Misclassified node (Node 1894)

Figure 6: Explainer results for nodes with highest betweenness centrality.

4 Link Prediction

4.1 Data Preprocessing

To prepare the data for the link prediction task, I employed a random link split strategy. I divided the edges into train (80% of the links), validation and test sets (10% each). For each training epoch, new negative edges were sampled to

ensure that the model was exposed to varied examples, while validation and test sets were of course left unchanged. The proportion of positive to negative edges was consistently maintained at 50% to balance the training dataset and prevent bias in the learning process.

4.2 Models

For the link prediction task, I tested the same architectures as in the previous experiments, with two different output modes regulated by an additional hyperparameter. In the first mode, the output consisted of the dot product of the embeddings of the two nodes. In the second mode, a final Multilayer Perceptron (MLP) was trained on the concatenation of the embeddings of the two nodes to produce the final prediction.

In addition, for the Node2Vec model, I conducted a series of experiments exploring different strategies. Both the dot product and logistic regression trained on the concatenated embeddings yielded acceptable results. However, I discovered that training a logistic regression model on the Hadamard product (element-wise multiplication) of the node embeddings completely solved the task, achieving excellent performance.

Inspired by the success of this approach with Node2Vec, I experimented with the Hadamard product for my graph neural network models. This decision was based on the observation that concatenating embeddings consistently underperformed compared to the dot product in earlier experiments. Additionally, I replaced the initially implemented complex MLP architecture with a simpler logistic regression model. As a result, the SAGE network ultimately selected this alternative over the dot product during hyperparameter tuning.

4.3 Results

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
GCN	0.7579	0.6841	0.9585	0.7984	0.9022
SAGE	0.7095	0.7512	0.6265	0.6832	0.7819
GAT	0.7362	0.6695	0.9328	0.7795	0.8777
DeepGCN	0.7302	0.6736	0.8933	0.7681	0.8347
DeepSAGE	0.7747	0.7158	0.9111	0.8017	0.8597
DeepGAT	0.7579	0.6998	0.9032	0.7886	0.8501
Node2Vec	**0.9921**	**0.9844**	**1.0000**	**0.9922**	**0.9993**

Table 4: Performance Metrics on Link Prediction Task for Cora Dataset

Node2Vec stands out as the most effective model for link prediction, achieving the highest metrics across the board. Its ability to capture structural equivalences and neighborhood proximity makes it particularly well-suited for this task. The model's ability to learn a low-dimensional representation of nodes

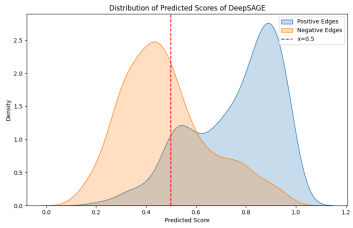
based on random walks and embeddings contributes to its superior performance in predicting links.

DeepGCN, DeepSAGE, and DeepGAT also demonstrate competitive performance, with balanced F1-scores and high ROC-AUC values. These models leverage advanced graph neural network architectures, capturing both node features and edge information. Notably, these models often outperform their 2-layer counterparts, highlighting the advantages of slightly deeper architectures in capturing more complex relationships between nodes.

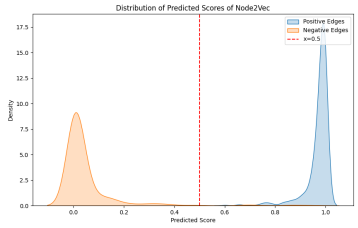
It is noticeable that the SAGE architecture, using MLP instead of the dot product for embedding aggregation, achieves the highest precision among all models. However, this comes at the cost of lower recall and accuracy.

All the models in my experiments were trained using the dot product for link prediction and always consisted of either 2 or 3 layers. These models generally excel at identifying positive edges, as indicated by their strong recall scores, accurately predicting when a link exists between two nodes. However, they tend to struggle with negative edges, resulting in higher false positive rates and lower precision in some cases.

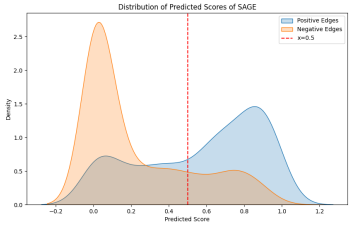
In contrast, the SAGE architecture exhibits the opposite behavior, performing better on negative edges than positive ones. This emphasizes how altering the final layer of the architecture can deeply influence the model's behavior and performance.



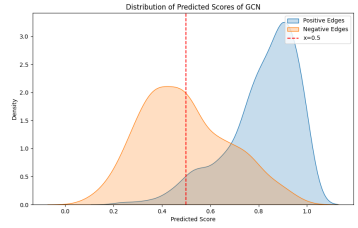
(a) Score distributions of DeepSAGE



(b) Score distributions of Node2Vec



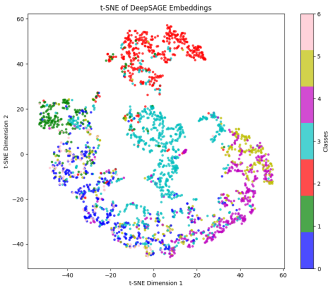
(c) Score distributions of SAGE



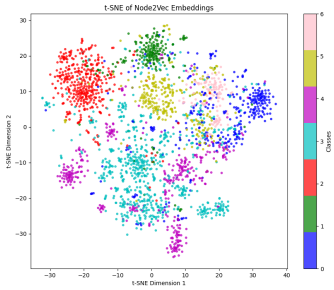
(d) Score distributions of GCN

Figure 7: Score distributions of the models

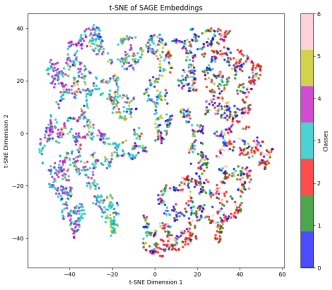
4.4 Embedding and Node2Vec comparison



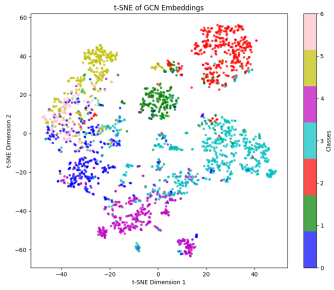
(a) Embeddings of DeepSAGE



(b) Embeddings of Node2Vec



(c) Embeddings of SAGE



(d) Embeddings of GCN

Figure 8: Node embeddings comparison using t-SNE

The embeddings generated by the 2-layer models show a clear separation of nodes into distinct classes, similar to what is observed in node classification tasks. At the same time, these embeddings capture more complex relationships between nodes.

In deeper architectures, the embeddings become more intricate, capturing more interactions between nodes, while still maintaining class separation.

The embeddings produced by the SAGE model, however, are qualitatively different from those of the other models, reflecting the unique properties of its MLP aggregation mechanism. This architectural difference results in a distinct embedding structure, which is decoded through the MLP. Additionally, the increased sparsity of its embeddings could contribute to its relatively poor performance in predicting positive edges.

In contrast, Node2Vec embeddings remain largely consistent with those from previous experiments, as the model's training setup was unchanged, with the only modification being the structure of the final logistic regression layer.

4.5 Analysis

The ability to predict links is crucial for numerous graph-based applications. A relevant example is recommender systems, where nodes represent users and items, and links indicate interactions, such as a user purchasing a specific item. By leveraging user/item features and the graph structure, link prediction models can infer new interactions, enabling personalized recommendations. This approach captures user preferences and reveals hidden patterns in the graph, improving recommendation accuracy.

In this case, Node2Vec is a highly robust choice for link prediction, as it excels at capturing graph patterns and clustering users and items based on their structural relationships within the graph. By focusing on the graph's connectivity and proximity, Node2Vec can identify potential links with remarkable accuracy, making it suitable for this kind of systems.

However, in environments where users or items are associated with meaningful features—such as item descriptions, keywords, or user preferences, as seen in e-commerce or movie recommendation systems, Graph Convolutional Networks (GCNs) or similar GNN-based architectures may be more appropriate. These models integrate both the graph's structural information and the feature data, creating a comprehensive representation that enhances personalized recommendations. Combining graph patterns with feature-rich embeddings could provide better performances.

For knowledge graph completion, where the goal is to predict missing links or relationships in a graph (e.g., filling in missing facts about entities), Node2Vec is indeed a strong choice due to its ability to capture both local and global graph structures through random walks. It can learn rich embeddings for entities (nodes) and relationships (edges), making it effective at predicting missing links in the graph. Moreover, knowledge graphs are often represented using the standard RDF graph model, where both entities and relationships are modeled as nodes and edges. This structure inherently embeds all the relevant information within the graph itself, without requiring any additional feature.

For social network analysis, the features of nodes often play a critical role in link prediction tasks. Social networks naturally contain rich information about users, such as demographics, preferences, or past behaviors, which can complement the structural relationships within the graph. In these cases, GNN-based models, such as GCNs or Graph Attention Networks (GATs), are particularly well-suited as they can effectively combine node features with graph structure to predict links. The incorporation of features allows these models to generalize better and provide insights into the dynamics of social interactions.

To conclude, GNN-based models demonstrate the capacity to incorporate additional features alongside the graph's structural information. This ability becomes particularly crucial in large-scale graphs, where patterns alone may fail to provide meaningful insights. By leveraging both node features and graph structure, GNNs are well-suited for scenarios where the complexity of the graph requires a more sophisticated understanding to predict links effectively.

On the other hand, Node2Vec excels in graphs where the connections be-

tween nodes encode the majority of the relevant information. Its ability to capture structural relationships and proximity through random walks makes it highly effective for link prediction in well-connected graphs. Together, these approaches highlight the importance of selecting the appropriate model based on the specific characteristics and requirements of the graph at hand.