SURVEY OF GRAPH NEURAL NETWORK MODELS ON DERGIPARK SCIENTIFIC ARTICLE DATASET

Gökay Gülsoy, Yusuf Emre Beskan
Department of Computer Engineering
İzmir Institute of Technology
İzmir, Gülbahçe 35433, Turkey
{gokaygulsoy,Yusufbeskan}@iyte.edu.tr

ABSTRACT

The rapid publication of thousands of research studies has made it increasingly difficult to track newly published research, particularly studies translated from English to Turkish, and to find research papers that belong to similar fields or are semantically related. This requires formal techniques to assess the classification of articles according to the field, topic, and content on which an article is written. The relationships among the research articles are usually many to many, which means that any article can be related to many other articles in terms of its research topic and content. In order to build a correct recommendation systems for such scenarios we can utilize state-of-the-art neural network models called Graph Neural Networks (GNNs). In our work we present evaluation of different GNN architectures and hyperparameters on 1040 article english-turkish scientific article dataset for node classification, node clustering, and link prediction tasks on graph structured data.

1 INTRODUCTION

As a part of our experimental study We have constructed 19 different Graph Neural Network models using PyTorch Geometric (PyG) (Fey & Lenssen, 2019) to asses the performance of different architectures and hyperparameters for three different graph learning downstream tasks which are node classification, node clustering, and link prediction respectively. Many of the neural network models existing are oriented towards structured data. For example, feed-forward neural networks for tabular data (Omondiagbe et al., 2019), convolutional neural networks for grid structured data such as images (Sathiyapriya & Shanthi, 2022), recurrent neural networks for sequential data such as text (Hu et al., 2020). In contrast to these data models, our problem is matching with graph data structure where each node can be related to any number of other nodes in which each node represents a single scientific research paper. Graph data can be provided to specialized neural network models called Graph Neural Networks (GNNs) for common downstream tasks such as classification (Kipf & Welling, 2017), clustering (Eddin et al., 2024), link prediction (Kipf & Welling, 2016), etc. to create efficient recommendation systems for research articles.

In our work firstly, We have created a graph dataset of 1040 english-turkish articles belonging to 5 different categories collected from DergiPark which is a service that provides the opportunity to publish national academic journals in electronic environment, hosting and process management, published by TÜBİTAK ULAKBİM. Secondly, we have constructed different GNN models by utilizing PyTorch Geometric (PyG) (Fey & Lenssen, 2019) which is a library built upon PyTorch to easily write and train Graph Neural Networks (GNNs) for a wide range of applications related to graph structured data. We have performed three downstream tasks two of which are node level which are node classification and node clustering, and one of which is edge level which is link prediction. Finally we have provided experimental results as evaluation metrics providing comparison of different Graph Neural Network models and architectures for three downstream task on our research for english-turkish article dataset. We have used BeautifulSoup library of Python to scrap english-turkish articles. We have collected articles for 5 different categories each category with following links and number of articles: 203 articles for Computer Science category from https://dergipark.org.tr/tr/pub/bbmd/archive and https://dergipark.

Table 1: Number of Articles in Each Category

CATEGORY	NUMBER OF ARTICLES
Computer Science	203
Social Sciences	210
Health Sciences	210
Environmental Sciences	207
Sports Sciences	210

org.tr/tr/pub/bbd/archive. 210 articles for Social Sciences category from https://dergipark.org.tr/tr/pub/asbi/archive. 210 articles for Health Sciences category from https://dergipark.org.tr/tr/pub/eujhs/archive. 207 articles for Environmental Sciences category from https://dergipark.org.tr/tr/pub/aucevrebilim/archive and https://dergipark.org.tr/tr/pub/ucbad/archive. 210 articles for Sports Sciences from https://dergipark.org.tr/tr/pub/jssr/archive. Our dataset of english-turkish articles is a balanced dataset such that each category has approximately same number of articles as given in table 1.

2 Related Works

Real challenge in most of the Graph ML tasks come from three fundamental problems, namely: scalability, diversity of domains on which Graph ML model to be developed, and levels at which predictions are going to be made such as node, link, or graph level (Hu et al., 2021). Firstly, in terms of scalability, most of the graph datasets used in benchmarking are very small in proportion to graphs encountered in many real-life applications which contain nodes more than 1 million in most cases (Bhatia et al., 2016; Husain et al., 2020; Vrandečić & Krötzsch, 2014). For instance, popular node-classification datasets such as PUBMED, CORA, and CITESEER have 2700 to 20,000 nodes (Yang et al., 2016), and commonly used knowledge graph datasets which are FB15k and WN18 restricted by 15,000 to 40,000 entities (Bordes et al., 2013). Most of the models developed by using these relatively small graph datasets which makes them not scalable to graphs that are greater in scale (Kipf & Welling, 2017; Trouillon et al., 2016).

Also there is no collective experimental approach followed by distinct studies. Separate studies use their own evaluation metrics and data-splitting strategies, which makes it harder to compare and contrast the performance of the models used in those studies (Dwivedi et al., 2023). As demonstrated by the work Hu et al. (2021) which provides realistic benchmark datasets to overcome these aforementioned issues. OPEN GRAPH BENCHMARK (OGB) poses three characteristics. First one is being large scale, OGB datasets are larger than other existing benchmarks and provided in three different scales (small, medium, and large). Second, coverage of diverse domains, OGB datasets target to include graphs which represent various domains such as molecular graphs, knowledge graphs, social networks, biological networks, etc. Third, is capability of handling different task categories, OGB supports making predictions at different graph levels which are at the level of node, link, and whole graph, respectively. At the time of writing of the paper Hu et al. (2021), OGB was supporting 15 datasets in which each category contained at least 4 datasets.

Research and studies on GNNs were mainly encouraged by achievements of CNNs in computer vision field in which the notion of convolution for image data was employed, then the idea of convolution was extended to graph data (Wu et al., 2021). Historically, graph kernel methods have been the leading techniques for solving graph classification tasks and use specialized functions called kernel functions to compute the similarity between two graphs (Kriege et al., 2020; Shervashidze et al., 2011). Kernel-based methods are computationally costly, whereas GNNs make use of extracted embeddings to perform classification, and thus are significantly more efficient than kernel methods for graphs (Kriege et al., 2020).

Wu et al. (2021) presents a taxonomy of GNNs by categorizing them into four main architectures, which are RecGNNs (recurrent graph neural networks), ConvGNNs (convolutional graph neural networks), GAEs (graph autoencoders), STGNNs (spatial-temporal neural networks), respectively.

RecGNNs are one of the first works of GNNs. They make use of recursive neural architectures in which each node constantly sends and receives messages to/from neighbors until a stable steady state is reached (Scarselli et al., 2009). ConvGNNs generalize the operation of convolution from grids of data such as images to graph data, this is done via stacking many layers of graph convolution layers and generating each node's high-level representation by aggregating it's own features and features of neighbors (Henaff et al., 2015). GAEs are unsupervised learning models that are capable of encoding nodes or graphs into latent vector spaces and then rebuild the graph data from encoded information in order to learn network embeddings and to generate graphs (Cao et al., 2016; Li et al., 2018).

Wu et al. (2021) provides experimental results for evaluation of node and graph classification tasks to assess performance of RecGNNs and ConvGNNs, but it does not provide rigorous and fair comparison. Shchur et al. (2019) points out two issues that cause unfair comparison of different GNN architectures. First, many of the proposed models have all been tested using same train/validation/test splits of three commonly used citation networks which are Cora, CiteSeer, and Pubmed. Problem with such an experimental setup is that it favors the model that overfits most and eliminates the actual purpose of using train/validation/test splits for which the main purpose is finding the model that has best generalization capability. Second, researchers often use training procedure that is quite different from the one they used for the baseline model which makes it hard to find out whether the performance improvement is a consequence of better architecture of new model or fairly well-tuned training procedure that unjustly utilizes new model. Also Melis et al. (2017) shows that simpler models often surpass sophisticated ones if hyperparameters are tuned equally carefully for all methods. Errica et al. (2020) provides a framework for rigorous model selection and assessment for GNNs, it is fair with respect to data splits and features assigned to competitor models, and also results are reproducible in contrast to most of the published results such as DiffPool (Ying et al., 2019), ECC (Simonovsky & Komodakis, 2017), GIN (Xu et al., 2019), and GraphSAGE (Hamilton et al., 2018) which depend on vague and lacking documentation.

3 METHODOLOGY

This section is divided into two subsections, subsection 3.1 discusses our methodology for dataset collection and generating data graph to be used in graph neural network (GNN) tasks. Subsection 3.2 provides an overview of the architecture and inner workings of graph neural networks we used in our experiments for node classification, node clustering, and link prediction. GitHub repository[1] provides implementation of the project, it can be easily extended to experiment with different GNN architectures and hyperparameters along with various graph ML downstream tasks.

3.1 DATA GRAPH GENERATION

Preliminary step for training our 19 different Graph Neural Network models is to create embeddings for nodes. We have generated node embedding for each node representing single article by using english abstract of article. In embedding generation using english abstracts we have used a pretrained sentence transformer model called SPECTER (Cohan et al., 2020) to generate 768 dimensional vector embeddings. Recent transformer models like BERT can learn powerful representations of texts , whereas these models are aiming towards sentence-level training purposes and do not make use of information on inter-document relatedness, this is a limiting feature for their capability to generate document-level representations (Devlin et al., 2019).

SPECTER is based on a transformer language model on a powerful signal of document-level relatedness which is citation graph and it does not require task-specific fine-tuning (Cohan et al., 2020). After embeddings are generated, We created an cosine similarity matrix to measure document relatedness and conducted comprehensive analysis of generated cosine similarity scores for each category and decided 0.73 as a threshold for connecting nodes that are semantically similar. Then We have generated an visualization of 768 dimensional english-articles by reducing them to 2 dimensional space with T-distributed Stochastic Neighbor Embedding (t-SNE) method (van der Maaten & Hinton, 2008) as given in figure 1. After creating edge list for our connected nodes, we used NetworkX library (Hagberg et al., 2008) to visualize our english-article based node embedding

[1]Link to Github project implementation

graph as given in figure 2. After obtaining english-article embeddings we have created an Pytorch tensor representing node features and another tensor representing our edge list. Then we have constructed Pytorch Geometric (PyG) Data object by providing node feature tensor then transposed and flattened edge list tensor as arguments respectively.

We used target variable as the category column of dataset and as each category is represented as a string in the dataframe's category column we used LabelEncoder class from sklearn to encode category target variable column into unique integers and then created a target tensor by providing encoded labels as a parameter. Different from usual machine learning tasks, we use train_test_split function from sklearn not directly with the dataset itself instead we create training, validation, and test masks by providing node ids as a data to split on. This approach is followed because we should only use a specific subset of nodes during training, validation, and testing where each subset must be disjoint in order to provide unbiased evaluation on validation and test sets. There are 29713 edges in the generated graph and figure 3 depicts number of nodes having specific node degree. Most of the nodes are are having degree in the range 0-50 as it can be seen in figure 3.

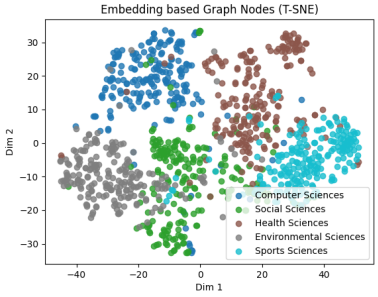

Figure 1: Node Embedding Visualization for English Abstracts with t-SNE

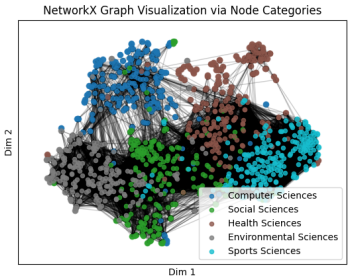

Figure 2: NetworkX Graph Visualization via Node Categories

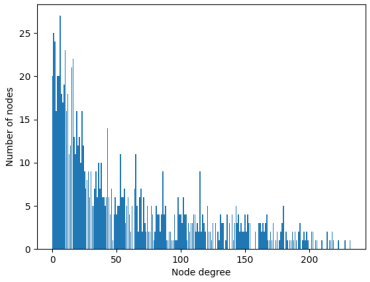

Figure 3: Node Numbers for each Node Degree

3.2 GRAPH NEURAL NETWORK MODELS USED IN EXPERIMENTS

In total we used 4 different graph neural network models (GNNs) which are Graph convolutional network (GCN) (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), Graph Attention Network (GAT) (Veličković et al., 2018), and Variational Graph Autoencoder (VGAE) (Kipf & Welling, 2016) respectively.

3.2.1 GRAPH CONVOLUTIONAL NETWORK (GCN)

GCN model is a graph neural network model that learns hidden layer representations that encode both the local structure of the graph and node features which can scale linearly with the number of edges in the graph. GCN architecture make use of label smoothing over the graph via some form of graph-based regularization utilizing Laplacian regularization term in the loss function as indicated by equation 1.

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}\,, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \left\| f\left(X_i\right) - f\left(X_j\right) \right\|^2 = f(X)^\top \Delta f(X) \tag{1}$$

In the eq. 1, $\mathcal{L}_0$ represents supervised loss with respect to labeled part of the graph, $f(.)$ can be a neural network-like differentiable function, λ is used as a weighing term and X indicates node feature matrix. $\Delta = D - A$ unnormalized Laplacian of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where N is the number of nodes and vertices $v_i \in V$, edges $(v_i, v_j) \in \mathcal{E}$, an adjacency matrix $A \in \mathbb{R}^{NxN}$ and a degree matrix $D_{ii} = \sum_j A_{ij}$. The formula indicated by the eq. 1 uses the assumption of connected nodes in the graph are more likely to share same label. Downside of this assumption is that graph edges are not required to encode similarity, they may provide some extra information instead. Critical point here is conditioning $f(.)$ on the adjacency matrix which allows model to distribute gradient information from supervised loss $\mathcal{L}_0$ and it enables model to learn representations of nodes which have labels and do not have labels (Kipf & Welling, 2017). GCN uses the layer-wise propagation rule indicated by eq. 2.

$$H^{(l+1)} = \sigma\left(\bar{D}^{-\frac{1}{2}} A \bar{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right) \tag{2}$$

In eq. 2 $\bar{A} = A + I_N$ is representing the adjacency matrix of the undirected graph $\mathcal{G}$ extended with the self-edges. I_N is the identity matrix, $\bar{D}_{ii} = \sum_j \bar{A}_{ij}$ and $W^{(l)}$ is a trainable weight matrix specific to each layer. $\sigma(.)$ denotes an activation function, such as $ReLU(.) = max(0, .)$. $H^{(l)}$ is the matrix of activations in the l^{th} layer; for the first layer $H^{(0)} = X$.

3.2.2 SPECTRAL CONVOLUTIONS WITH GRAPHS

Spectral convolutions on graphs are defined as the multiplication of signal $x \in \mathbb{R}^N$ with the filter $g_\theta = \text{diag}(\theta)$ taking a parameter $\theta \in \mathbb{R}^N$ in Fourier domain as indicated by eq. 3.

$$g_\theta * x = U g_\theta U^\top x \tag{3}$$

in which U denotes the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^\top$ with Λ being diagonal matrix of eigenvalues and $U^\top x$ being the graph Fourier transform of x. Evaluating eq. 3 is costly, because multiplication operation with eigenvector matrix U is $\mathcal{O}(N^2)$. Moreover, computation of eigenvalue decomposition of Laplacian L can be extremely expensive for large scale graphs. To overcome this problem $g_\theta(\Lambda)$ can be approximated to an extent via use of Chebyshev polynomials T_k up to K^{th} order term as indicated by eq. 4.

$$g_\theta(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\hat{\Lambda}) \tag{4}$$

$\tilde{\Lambda} = \frac{\lambda}{\lambda_{\max}}\Lambda - I_N$ is representing rescaled Λ. λ_{max} denotes the largest eigenvalue of L. $\theta' \in \mathbb{R}^{\mathbb{K}}$ is denoting vector of Chebyshev coefficients. The Chebyshev polynomials are recursively formulated as $T_k(x) = 2xT_{k-1}(x) - T_{K-2}(x)$, where $T_0(x) = 1$ and $T_1(x) = x$ (Hammond et al., 2009).

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x, \tag{5}$$

with $\tilde{L} = \frac{2}{\lambda_{\max}}L - I_N$ and $\left(U\Lambda U^\top\right)^k = U\Lambda^k U^\top$. That expression now represents K-localization since it is a K^{th}-order polynomial in the Laplacian, it depends only on nodes that are located K steps away from the central node (K^{th}-order neighborhood). The complexity of evaluating eq.5 is $\mathcal{O}(|\mathcal{E}|)$ linear in the number of edges and used to define convolutional neural network on graph data (Defferrard et al., 2017). Semi-supervised node classification on a graph is defined by the eq. 6 as given below:

$$Z = f(X, A) = \operatorname{softmax}\left(\hat{A}\,\mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) \tag{6}$$

First step is to compute $\hat{A} = \bar{D}^{-\frac{1}{2}}A\bar{D}^{-\frac{1}{2}}$ as a pre-processing step. $W^0 \in \mathbb{R}^{\mathbb{C}\mathrm{x}\mathbb{H}}$ is an input-to-hidden weight matrix for hidden layer with H feature maps and $W^1 \in \mathbb{R}^{\mathbb{H}\mathrm{x}\mathbb{F}}$ is hidden-to-output weight matrix. The softmax activation function is represented by $\operatorname{softmax}(x_i) = \frac{1}{\mathcal{Z}}exp(x_i)$ where $\mathcal{Z} = \sum_i exp(x_i)$ which is applied row-wise. Cross-entropy loss for multi-class classification is evaluated as given in the eq. 7.

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L}\sum_{f=1} Y_{lf} \ln Z_{lf} \tag{7}$$

in which Y_L denotes the set of node indices having labels. Weights of neural network $W^{(0)}$ and $W^{(1)}$ are trained with batched gradient descent using overall dataset for every training iteration. Using batched version over full dataset is valid if dataset fits in memory, because space complexity is $\mathcal{O}(|\mathcal{E}|)$ which is linear in the number of edges (Kipf & Welling, 2017).

3.2.3 GRAPHSAGE

Low-dimensional embeddings of nodes, especially in large scales graphs, allow many downstream machine learning tasks to be done computationally efficiently such as prediction, content recommendation, and biological molecule functioning (Hamilton et al., 2018). The fundamental idea behind node embedding approaches is to make use of dimensionality reduction methods to reduce the high-dimensional representation of node's neighborhood into dense vector embedding. However, most of the real-life applications require node embeddings to be constructed fast and generalize to unseen nodes, even to entirely new subgraphs. This is where inductive learning capability is a necessity for production grade machine learning systems that evolve constantly with many unseen nodes such as YouTube videos and Reddit posts (Hamilton et al., 2018). Such inductive node embedding problem is difficult in comparison to transductive approach because it requires an already optimized node embeddings to generalize and align with newly observed subgraphs.

Until the introduction of the GraphSAGE model, graph convolutional networks (GCNs) have only been applied to transductive learning scenarios in which graph structures are fixed (Kipf & Welling, 2017), (Kipf & Welling, 2016). GraphSAGE extends to inductive learning strategy and generalizes the GCN approach with trainable aggregation functions distinct from simply applying convolutions. GraphSAGE learns both the topological structure of each node's neighborhood and distribution of node features around the neighborhood. GraphSAGE uses aggregator functions which learn to aggregate information from node's neighborhood. Each aggregator aggregates information from various number of hops away from given node. Later, at inference time trained model can be used to produce embeddings for completely unseen nodes via applying learned aggregation functions. Algorithm 1 gives the pseudo-code of how GraphSAGE generates embeddings via forward propagation.

Algorithm 1 GraphSAGE embedding generation algorithm

Require: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{x_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $W^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions AGGREGATE$_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Ensure: Vector representations z_v for all $v \in \mathcal{V}$

$h_v^0 \leftarrow x_v, \forall v \in \mathcal{V}$
for $k = 1 \ldots K$ **do**
  for $v \in \mathcal{V}$ **do**
    $h_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})$
    $h_v^k \leftarrow \sigma\left(W^k \cdot \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k)\right)$
  end for
  for $v \in \mathcal{V}$ **do**
    $h_v^k \leftarrow \frac{h_v^k}{\|h_v^k\|_2}$
  end for
end for
for $v \in \mathcal{V}$ **do**
  $z_v \leftarrow h_v^K$
end for

The main idea behind the Algorithm 1 is that in each iteration nodes aggregate information from local neighborhood, as the iteration count increases, each node incrementally gain more information from further hops of the graph. Algorithm 1 explains the generation of node embeddings over the entire graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with features x_v, $\forall v \in \mathcal{V}$. Algorithm is described as follows, k denotes the current step of outer for loop which represent hop level and h^k represents node's embedding at current iteration step. Firstly, for each node $v \in \mathcal{V}$ aggregates the embeddings of nodes in its direct neighborhood, $\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}$, into a single vector $h_{\mathcal{N}(v)}^{k-1}$. So aggregation at each step depends on the embeddings generated at previous iteration of the outer loop, and for the first iteration where $k = 0$ ("base case") embeddings are assigned as node features. After neighborhood aggregation neighborhood vector $h_{\mathcal{N}}(v)^{k-1}$ is concatenated with the node's current embedding h_k^{k-1} and fed through a fully connected layer with an activation function σ, which produces embeddings to be used in the next iteration of the outer loop. Neighborhood embedding aggregation can be performed via different aggregator architectures (Hamilton et al., 2018).

Learning Parameters of GraphSAGE. GraphSAGE uses a graph-based loss function to output node embeddings, $z_u, \forall u \in \mathcal{V}$, and stochastic gradient descent is used to tune weight matrices, $\mathbf{W}^k, \forall k \in \{1, ..., K\}$ and parameters of aggregator functions. This loss function allows nodes that are close in proximity to have similar embeddings, while forces the nodes that are disparate to have highly distinct embeddings. Graph-based loss function is given in the eq. 8 below:

$$J_G(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right) \tag{8}$$

here v is a node that co-occurs with u on a fixed-length random walk, σ denotes the sigmoid function, P_n represents negative sampling distribution, and Q indicates number of negative samples. Rather than training unique embedding for each node used in matrix-factorization based or random-walk based methods (Cao et al., 2015), (Grover & Leskovec, 2016), node embeddings feed into loss function are generated from features aggregated around node's local neighborhood.

Aggregation Architectures. Three type of aggregators which are mean aggregator, LSTM aggregator and pooling aggregator are explained in the original GraphSAGE paper (Hamilton et al., 2017). Mean aggregator takes the element-wise mean of vectors in $\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}$. Mean aggregation operation is given in eq. 9 below:

$$\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W} \cdot \text{MEAN}\left(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}\right)\right) \tag{9}$$

The significant difference between the GraphSAGE aggregators and the convolutional aggregator is that the convolutional aggregator does not concatenate the embedding of the node from the previous

layer with the aggregated neighborhood embedding vector $h^k_{\mathcal{N}(v)}$. The LSTM-based aggregation operator uses an LSTM architecture (Hochreiter & Schmidhuber, 1997) to operate on a unordered set via applying LSTM to a random permutation of node's. neighbors. The last aggregator that is discussed is the pooling aggregator in which each embedding vector from node's immediate neighborhood is fed through a arbitrarily deep fully connected neural network (Charles et al., 2017) and element-wise maximum-pooling operation is applied to output of neural network in order to aggregate information from neighborhood set. The max-pooling aggregation operator is given in the equation. 10 below:

$$\text{AGGREGATE}^{\text{pool}}_k = \max\left(\left\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}^k_u + \mathbf{b}\right), \forall u \in \mathcal{N}(v)\right\}\right) \tag{10}$$

In eq. 10 $\max$ represents element-wise max operator and σ is an activation function. Multi-layer perceptron applied can be thought of as a set of functions used to compute features for each of the nodes embeddings in the local neighborhood.

3.2.4 Graph Attention Network (GAT)

Fundamental idea behind Graph Attention Network is that stacking of layers in which nodes are allowed to attend over their specific neighborhood features by providing different weights to distinct nodes in neighborhood (Veličković et al., 2018).

Graph Attention Layer. The input to Graph attention (GAT) layer is the set of node embeddings $h = \{\vec{h_1}, \vec{h_2}, ..., \vec{h_N}\}, \vec{h_i} \in \mathbb{R}^F$, where N denotes the number of nodes, and F represents the number of features for each node. Then Graph attention layer produces a new set of node embeddings $h' = \{\vec{h'_1}, \vec{h'_2}, ..., \vec{h'_N}\}$, $\vec{h'_i} \in \mathbb{R}^F$ where number of features is F' possibly having different cardinality. Initial step is to perform a linear transformation which uses a weight matrix $\mathbf{W} \in \mathbb{R}^{F' x F}$ applied to each node. Then *self-attention* mechanism, $a : \mathbb{R}^{\mathbb{F}'}$ x $\mathbb{R}^{\mathbb{F}'} \rightarrow \mathbb{R}$ is applied to each node in order to compute *attention-coefficients* as given in the eq. 11 below:

$$e_{ij} = a(\mathbf{W}\vec{h_i}, \mathbf{W}\vec{h_j}) \tag{11}$$

attention coefficient reflects the importance of features of node j to node i. Graph structure is injected to attention mechanism via using *masked-attention* such that e_{ij} is computed solely for the nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ represents local neighborhood of node i in the graph. After attention coefficients are computed, they are normalized across all the j via applying softmax function as given in eq. 12 below:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}. \tag{12}$$

Self-attention mechanism a is feed-forward neural network which is parametrized by weight, and applies the LeakyReLU activation function. Attention coefficients which are computed via attention mechanism is given in full form in eq. 13 below:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\vec{h}_i \,\|\, \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\vec{h}_i \,\|\, \mathbf{W}\vec{h}_k]\right)\right)} \tag{13}$$

$\|$ is the concatenation operation and $a.^T$ represents transposition of attention coefficients. For K number of attention heads, aggregation and node embedding generation $\forall v \in V$ is described in eq. 14 below:

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathcal{N}_i} \alpha^k_{ij}\mathbf{W}^k\vec{h}_j\right) \tag{14}$$

3.2.5 VARIATIONAL GRAPH AUTO-ENCODERS (VGAE)

Variational Graph Auto-encoder architecture provides a framework for unsupervised learning on graphs where the idea is originated from *variational auto-encoder* architectures (Kingma & Welling, 2013), (Rezende et al., 2014). For an unweighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with number of nodes $N = |V|$, adjacency matrix of the graph is indicated by $\mathbf{A}$, degree matrix is represented by $\mathbf{D}$, latent variables are represented as matrix $\mathbf{Z}$, and node features are represented by feature matrix $\mathbf{X}$. Variational graph auto-encoder model in the original paper makes use of two-layer GCN (Kingma & Welling, 2013) in which one convolution layer is used to learn mean of normal distribution, second convolution layer is used to learn standard deviation of normal distribution from which node embeddings are sampled. Eq. 15 describes sampling of node embeddings from a normal distribution.

$$q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}), \quad \text{with} \quad q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \text{diag}(\boldsymbol{\sigma}_i^2)) \tag{15}$$

In eq. 15, μ = $\text{GCN}_\mu(X, A)$ is representing matrix of mean vectors, and $\log \sigma$ = $\text{GCN}_\sigma(X, A)$ represents the matrix of standard deviation vectors. Two-layer GCN is expressed as GCN(X,A) = $\tilde{A}\text{ReLU}(\tilde{A}XW_0)W_1$, along with weight matrices W_i. μ = $\text{GCN}_\mu(X, A)$ and σ = $\text{GCN}_\sigma(X, A)$ share the initial-layer weights W_0.

Generative model Generative model implemented as a inner product between sampled embeddings as indicated by eq. 16 below:

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j), \quad \text{with} \quad p(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j) \tag{16}$$

Loss function Loss function is defined in eq. 17 below and it is optimized to minimize the loss:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})} \left[\log p(\mathbf{A} \mid \mathbf{Z})\right] - \text{KL}\left[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \,\|\, p(\mathbf{Z})\right] \tag{17}$$

In which KL[$q(.)||p(.)$] is the term defined as Kullback-Leibler divergence between $q(.)$ and $p(.)$. In non-probabilistic version of VGAE model, embeddings $\mathbf{Z}$ and reconstructed adjacency matrix $\hat{A}$ computed as in eq. 18 below:

$$\hat{A} = \sigma(ZZ^\top), \quad Z = GCN(\mathbf{X}, \mathbf{A}) \tag{18}$$

4 EVALUATION

In this section We discuss different evaluation metrics which are accuracy, precision, recall, F1-score, normalized mutual information, adjusted random index, area under ROC curve (AUC), and average precision (AP) that We have used for node classification, node clustering, and link prediction tasks in our experiments.

4.1 METRICS USED IN NODE CLASSIFICATION

Accuracy Accuracy measure tells the number of correctly classified instances by classification model. It is of two kind: training accuracy and testing accuracy and they have no direct correlation to each other (Bansal & Singhrova, 2021). Eq. 19 describes accuracy as below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{19}$$

In eq. 19 for accuracy above TP, TN, FP, FN represents true positives, true negatives, false positives, and false negatives respectively.

Testing Error Testing error is observed when the model is trained and then testing is performed on test dataset (Bansal & Singhrova, 2021).

Precision Precision is the ratio of correctly predicted positives to everything classified as positive (Bansal & Singhrova, 2021). Eq. 20 describes precision as below:

$$Precision = \frac{TP}{TP + FP} \tag{20}$$

Recall Recall which is also known as True positive Rate (TPR) is the ratio of correctly predicted positives to all instances that are actually positive (Bansal & Singhrova, 2021). Eq. 21 describes recall as below:

$$Recall = \frac{TP}{TP + FN} \tag{21}$$

F1-Score F1-Score is defined as the harmonic mean of precision and recall. It favors the algorithms having high susceptibility and also known as F-score or F-measure (Bansal & Singhrova, 2021). Eq. 22 describes F1-score as below:

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{22}$$

4.2 Metrics Used in Node Clustering

Mutual Information Mutual Information can be defined in terms of information theory as the amount of information required to transmit a labelling from sender to receiver (Jerdee et al., 2024). Mutual Information between ground-truth labellings which we denote with g, and candidate labellings denoted as c is defined as given in eq. 23 below:

$$I(c; g) = H(g) - H(g|c) \tag{23}$$

Mutual information is defined as the total entropy of g minus the conditional entropy of g given c where $H(x)$ represents the entropy of x and $H(x|y)$ represents the entropy of x given y. Difficulty of measuring similarity with Mutual information is that range of values depend on specific application making it difficult to decide when a score is large or small. To solve this problem, Mutual information score is normalized so that maximum value it can take is 1 in case all the labels in candidate are agreeing with ground truth labellings (Jerdee et al., 2024).

Normalized Mutual Information Normalized Mutual information (NMI) makes use of Mutual Information $I_0(c; g)$ as it's base measure and performs normalization as given in eq. 24 below:

$$\begin{aligned} NMI_0^{(S)} &= \frac{I_0(c; g)}{\frac{1}{2}[H_0(c) + H_0(g)]} \\ &= \frac{I_0(c; g)}{\frac{1}{2}[H_0(c) + H_0(g)]} \end{aligned} \tag{24}$$

Random Index Random Index is one of the performance indices used for cluster evaluation. Indices are used as measures of correspondence between two partitions of same dataset and rely on how pairs of instances are classified in contingency table (Santos & Embrechts, 2009).

More formally, consider a set of n objects $S = \{O_1, O_2, ..., O_n\}$ and two partitions $U = \{u_1, u_2, ..., u_R\}$ and $V = \{v_1, v_2, ..., v_R\}$ of objects in S such that $\bigcup_{i=1}^{R} u_i = S = \bigcup_{i=1}^{C} v_j$ and $u_i \cap u_{i'} = \emptyset = v_j \cap v_{j'}$ for $1 \leq i \neq i' \leq R$ and $1 \leq j \neq j' \leq C$.

From a given set, results form four different groups for pairs as follows:

a - objects in pair are located in the same group in U and in the same group in V.

b - objects in pair are located in the same group in U and in different groups in V.

c - objects in pair are located in the same group in V and in different groups in U.

d - objects in pair are located in the different groups in U and in different groups in V.

Contingency table for partitions is given in the table 2 below:

Table 2: Contingency Table for Comparing Partitions U and V.

Partition	V	
U	Pair in same group	Pair in different groups
Pair in same group	a	b
Pair in different groups	c	d

Random Index proposed by Rand (and, 1971) is computed as in eq. 25 below:

$$RI = \frac{a+d}{a+b+c+d} \tag{25}$$

Adjusted Random Index Whereas there are some known problems with Random Index (RI) method such as Random Index of two partitions cannot take constant value or Rand statistics converges to upper limit of unity as the number of clusters increases progressively (Santos & Embrechts, 2009). In order to overcome these problems Adjusted Random Index is proposed by researchers as an improvement (Hubert & Arabie, 1985) over Random Index. It is the recommended choice of index for measuring agreement between two partitions in clustering analysis with various numbers of clusters. Adjusted Random Index having expected value of 0 and maximum value of 1 is computed as given in eq. 26 below:

$$ARI = \frac{\binom{n}{2}(a+d) - [(a+b)(a+c)+(c+d)(b+d)]}{\binom{n}{2}^2 - [(a+b)(a+c)+(c+d)(b+d)]} \tag{26}$$

4.3 Metrics Used in Link Prediction

Area Under ROC Curve (AUC) Let i denote the *ordered* index, that is $x_1 \geq x_2 \geq ... \geq x_n$. If the scores are threshold at x_k, assigning all those instances with scores $\geq x_k$ to be class-1 and all others having scores $\leq x_k$ to be class-0. Also let $d(k)$ be the number of subjects having scores $\geq x_k$, and $\tilde{h}(k)$ be the number of subjects truly belonging to class-1 out of all those class-1 instances, and $n1$ be represent the total number of instances that are belonging to class-1.

Confusion matrices are commonly represented in terms of proportions rather than counts as given in the eq. 27 below:

$$t \equiv \frac{d(k)}{n}, \quad \pi \equiv \frac{n_1}{n}, \quad \text{and} \quad h(t) \equiv \frac{\tilde{h}(k)}{n}. \tag{27}$$

The collection of points, $\{(t, h(t), t \in [0,1]\}$, forms a so-called *hit curve*.

Proposition 1. *Let $h(t)$ be a hit curve which is continuous and differentiable almost everywhere. Then,*

(a) $h(0) = 0$ *and* $h(1) = \pi$*;*

(b) $0 \leq h'(t) \leq 1$*, for all t;*

(c) $\int h(t)\, dh(t) = \pi^2/2$.

Eq. 28 below defines the *true positive fraction* (TPF) and *false positive fraction* (FPF). ROC curve is defined as the collection of points $\{(\text{FPF}(t), \text{TPF}(t), t \in [0, 1]\}$.

$$\begin{aligned} \text{TPF}(t) &= \frac{h(t)}{\pi} \quad \left(\text{or} \quad \text{TPF}(k) = \frac{\tilde{h}(k)}{n_1} \right) \\ \text{FPF}(t) &= \frac{t - h(t)}{1 - \pi} \quad \left(\text{or} \quad \text{FPF}(k) = \frac{d(k) - \tilde{h}(k)}{n - n_1} \right) \end{aligned} \tag{28}$$

Area under the ROC curve (AUC) is defined as in terms of TPF(t) FPF(t) as given in the following integral below in eq. 29:

$$\begin{aligned} \text{AUC} &= \int \frac{h(t)}{\pi} \, d \left[\frac{t - h(t)}{1 - \pi} \right] \\ &= \frac{1}{\pi(1 - \pi)} \left[h(t) \left(dt - dh(t) \right) \right] \\ &= \frac{1}{\pi(1 - \pi)} \left[\int h(t) \, dt - \int h(t) \, dh(t) \right] \\ &= \frac{1}{\pi(1 - \pi)} \left[\int h(t) \, dt - \frac{\pi^2}{2} \right], \end{aligned} \tag{29}$$

Final replacement in eq. 29 is performed according to item (c) of proposition 1.

Average Precision Precision and Recall metrics as are defined as in eq. 30 and eq. 31 below (Su et al., 2013):

$$Precision(t) = \frac{h(t)}{t} \left(or \quad Precision(k) = \frac{h(k)}{d(k)} \right) \tag{30}$$

$$Recall(t) = \frac{h(t)}{\pi} \left(or \quad Recall(k) = \frac{h(k)}{n_1} \right) \tag{31}$$

The average precision (AP) is described as closed form in eq. 32 (Zhu, 2004) and expanded form of AP is described in eq. 33:

$$\int Precision(t) d[Recall(t)] \tag{32}$$

$$\begin{aligned} AP &= \int \frac{h(t)}{t} \times d \frac{h(t)}{\pi} \\ &= \frac{1}{\pi} \int \frac{h(t)}{t} dh(t) \end{aligned} \tag{33}$$

4.4 Graph Neural Network Model Details Used In Experiments

We have used 19 different graph neural network models in total where 18 of these models are used in node classification and node clustering, 1 model is used in link prediction. All the models used in experiments are listed as below:

1. **Model-1** GCN with 2 Convolution Layers
2. **Model-2** GCN with 2 Convolution Layers and 0.5 dropout ratio
3. **Model-3** GCN with 3 Convolution Layers

4. **Model-4** GCN with 2 Convolution Layers and 0.8 dropout ratio
5. **Model-5** GCN with 3 Convolution Layers and 0.5 dropout ratio
6. **Model-6** GraphSAGE with 2 Convolution Layers with mean aggregation
7. **Model-7** GraphSAGE with 2 Convolution Layers with mean aggregation and 0.5 dropout ratio
8. **Model-8** GraphSAGE with 2 Convolution Layers with max aggregation and 0.5 dropout ratio
9. **Model-9** GraphSAGE with 3 Convolution Layers with mean aggregation and 0.5 dropout ratio
10. **Model-10** GraphSAGE with 3 Convolution Layers with min aggregation and 0.5 dropout ratio
11. **Model-11** GraphSAGE with 3 Convolution Layers with mean aggregation and 0.5 dropout ratio
12. **Model-12** GraphSAGE with 2 Convolution Layers with Softmax aggregation with learnable parameter t and 0.5 dropout ratio
13. **Model-13** GraphSAGE with 3 Convolution Layers with Standard Deviation aggregation and 0.5 dropout ratio
14. **Model-14** GAT with 2 Convolution Layers with 1 attention head
15. **Model-15** GAT with 2 Convolution Layers with 2 attention heads and 0.5 dropout ratio
16. **Model-16** GAT with 2 Convolution Layers with 1 attention head and 0.5 dropout ratio
17. **Model-17** GAT with 3 Convolution Layers with 1 attention head and 0.5 dropout ratio
18. **Model-18** GCN with 2 Convolution Layers with 8 attention heads and 0.5 dropout ratio
19. **Model-19** VGAE with 3 Convolution Layers

4.5 Experimental Results

This section gives the numerical results acquired by evaluating different graph neural network architectures and models using the metrics defined under the sections 4.1, 4.2, 4.3 on node classification, node clustering, and link prediction tasks. All models are configured to use Adam optimizer (Kingma & Ba, 2017) and learning rate of 0.01 and fine-tuned in terms of the number of training epochs to be able to observe performance differences originating mainly from differences in model architecture.

4.5.1 Node Classification Experimental Results

Table 3 given below gives the numerical results for test loss, test accuracy, F1 score, precision, and recall metrics for the node classification task.

Table 3: Node classification results for different model architectures.

Model	Test Loss	Accuracy	F1-score	Precision	Recall	Epochs
Model-1	0.3415	91.28%	0.9128	0.9128	0.9128	200
Model-2	0.2406	91.75%	0.9175	0.9175	0.9175	200
Model-3	0.2923	89.09%	0.8909	0.8909	0.8909	180
Model-4	**0.2023**	**93.53%**	**0.9353**	**0.9353**	**0.9353**	180
Model-5	0.4863	90.86%	0.9086	0.9086	0.9086	180
Model-6	0.2563	91.12%	0.9112	0.9112	0.9112	40
Model-7	0.3405	89.83%	0.8983	0.8983	0.8983	60
Model-8	0.4304	87.87%	0.8787	0.8787	0.8787	50
Model-9	0.3087	92.55%	0.9255	0.9255	0.9255	200
Model-10	0.3585	91.18%	0.9118	0.9118	0.9118	50
Model-11	0.3015	91.08%	0.9108	0.9108	0.9108	300

Table 3 – continued from previous page

Model	Test Loss	Accuracy	F1-score	Precision	Recall	Epochs
Model-12	0.2743	90.80%	0.9080	0.9080	0.9080	200
Model-13	0.3477	91.47%	0.9147	0.9147	0.9147	200
Model-14	0.3898	86.96%	0.8696	0.8696	0.8696	40
Model-15	0.2190	91.67%	0.9167	0.9167	0.9167	300
Model-16	0.2670	91.36%	0.9136	0.9136	0.9136	300
Model-17	0.4043	88.04%	0.8804	0.8804	0.8804	300
Model-18	0.3094	89.50%	0.8950	0.8950	0.8950	300

4.5.2 NODE CLUSTERING EXPERIMENTAL RESULTS

Table 4 gives numerical results for Normalized Mutual Information (NMI) and Adjusted Random Index (ARI) metrics for the node clustering task as given below:

Table 4: Node clustering results for different model architectures.

Model	NMI	ARI	Epochs
Model-1	0.4458	0.3085	200
Model-2	0.4804	0.3539	200
Model-3	0.3867	0.2267	200
Model-4	0.4819	0.3490	200
Model-5	0.4102	0.2606	200
Model-6	**0.6786**	0.5906	300
Model-7	0.5358	0.3831	300
Model-8	0.6173	0.4586	300
Model-9	0.3517	0.1728	300
Model-10	0.4070	0.2290	300
Model-11	0.3810	0.2099	300
Model-12	0.5260	0.3504	300
Model-13	0.5532	0.4262	40
Model-14	0.6777	**0.6257**	200
Model-15	0.6020	0.4927	200
Model-16	0.6696	0.5789	200
Model-17	0.6150	0.5307	200
Model-18	0.5332	0.3905	200

4.5.3 LINK PREDICTION EXPERIMENTAL RESULTS

Table 5 gives numerical result for Area Under ROC curve (AUC) and Average Precision (AP) metrics for the link prediction task as given below:

Table 5: Link prediction result for Variational Graph Auto-Encoder model.

Model	AUC	AP	Epochs
Model-19	0.9316	0.9349	200

4.5.4 INTERPRETING EXPERIMENTAL RESULTS

According to results obtained by running experiments, on node classification task lowest test loss that is **0.2023** and highest test accuracy **93.53%** is achieved by **Model-4**. On node clustering task, highest NMI values is achieved by **Model-6** with NMI score of **0.6786**, and highest ARI score is achieved by **Model-14** with ARI score of **0.6257**. Consequently, GCN model performed best on node classification, GraphSAGE model reached highest NMI score on node clustering, and GAT

model reached highest ARI score on node clustering. VGAE model for linked prediction reached **0.9316** as AUC score and **0.9349** as AP score.

5 CONCLUSION

This paper presents a framework for comparing performance of 19 different graph neural network models for fundamental graph machine learning tasks which are node classification, node clustering, and link prediction respectively. Graph creation is done from scratch via collecting 1040 raw article data from DergiPark, cleaning and preprocessing article data using Pandas library and turning cleaned article data into graph data structure with the help of Pytorch Geometric (PyG) library Fey & Lenssen (2019). NetworkX library Hagberg et al. (2008) was used to visualize graph dataset and edges among different article categories. GCN, GraphSAGE, and GAT GNN architectures are used in node classification and node clustering tasks and VGAE architecture used in link prediction task.

GCN based model performed best in node classification, in node clustering task GraphSAGE based model reached highest NMI score and GAT based model reached highest ARI score. VGAE model for link prediction produced promising results with high AUC and AP scores. In conclusion, this study provides a complete end-to-end Graph ML pipeline starting with the english-turkish article data set collection, cleaning and preprocessing article data set, creating graph from article dataset with respect to similarity scores obtained after generating embeddings for english articles with SPECTER sentence encoder model Cohan et al. (2020), creating GCN, GraphSAGE, GAT, and VGAE GNN models with PyG libraryHagberg et al. (2008), fine-tuning each model by running appropriate number of epochs via experimentation, and finally evaluating node classification, node clustering, and link prediction tasks with appropriate metrics and reporting best performing model for each task. Future works can be extending fundamental graph ML downstream tasks (node classification, node clustering, link prediction) to larger and more diverse data sets, experimenting with GNN architectures apart from GCN, GraphSAGE, GAT, and VGAE as proposed in literature, enhancing model performances with integration of Retrieval-augmented-generation (RAG) with knowledge graphs.

REFERENCES

William M. Rand and. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. doi: 10.1080/01621459.1971.10482356. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356.

Aayushi Bansal and Anita Singhrova. Performance analysis of supervised machine learning algorithms for diabetes and breast cancer dataset. In *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, pp. 137–143, 2021. doi: 10.1109/ICAIS50930.2021.9396043.

K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016. URL http://manikvarma.org/downloads/XC/XMLRepository.html.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015. URL https://api.semanticscholar.org/CorpusID:17341970.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016. doi: 10.

1609/aaai.v30i1.10179. URL https://ojs.aaai.org/index.php/AAAI/article/view/10179.

R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 77–85, 2017. doi: 10.1109/CVPR.2017.16.

Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S. Weld. Specter: Document-level representation learning using citation-informed transformers, 2020. URL https://arxiv.org/abs/2004.07180.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017. URL https://arxiv.org/abs/1606.09375.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *J. Mach. Learn. Res.*, 24(1), January 2023. ISSN 1532-4435.

Maymouna Ez Eddin, Mohamed Massaoudi, Haitham Abu-Rub, Mohammad Shadmand, and Mohamed Abdallah. Graph neural network-based node clustering for dual-focused power network partitioning. In *IECON 2024 - 50th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1–6, 2024. doi: 10.1109/IECON55916.2024.10905473.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HygDF6NFPB.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019. URL https://arxiv.org/abs/1903.02428.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016. URL https://arxiv.org/abs/1607.00653.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman (eds.), *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15, Pasadena, CA USA, 2008.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. URL https://arxiv.org/abs/1706.02216.

David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory, 2009. URL https://arxiv.org/abs/0912.3848.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data, 2015. URL https://arxiv.org/abs/1506.05163.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Haojin Hu, Mengfan Liao, Chao Zhang, and Yanmei Jing. Text classification based recurrent neural network. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pp. 652–655, 2020. doi: 10.1109/ITOEC49072.2020.9141747.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs, 2021. URL https://arxiv.org/abs/2005.00687.

Lawrence J. Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985. URL https://api.semanticscholar.org/CorpusID:189915041.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search, 2020. URL https://arxiv.org/abs/1909.09436.

Maximilian Jerdee, Alec Kirkley, and M. E. J. Newman. Normalized mutual information is a biased measure for classification and community detection, 2024. URL https://arxiv.org/abs/2307.01282.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. URL https://api.semanticscholar.org/CorpusID:216078090.

Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016. URL https://arxiv.org/abs/1611.07308.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1), January 2020. ISSN 2364-8228. doi: 10.1007/s41109-019-0195-3. URL http://dx.doi.org/10.1007/s41109-019-0195-3.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs, 2018. URL https://arxiv.org/abs/1803.03324.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *ArXiv*, abs/1707.05589, 2017. URL https://api.semanticscholar.org/CorpusID:33513311.

David A. Omondiagbe, Shanmugam Veeramani, and Amandeep S. Sidhu. Machine learning classification techniques for breast cancer diagnosis. *IOP Conference Series: Materials Science and Engineering*, 495(1):012033, apr 2019. doi: 10.1088/1757-899X/495/1/012033. URL https://dx.doi.org/10.1088/1757-899X/495/1/012033.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/rezende14.html.

Jorge Santos and M. Embrechts. On the use of the adjusted rand index as a metric for evaluating supervised classification. volume 2009, pp. 175–184, 09 2009. ISBN 978-3-642-04276-8. doi: 10.1007/978-3-642-04277-5_18.

G. Sathiyapriya and S. Anita Shanthi. Image classification using convolutional neural network. In *2022 First International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, pp. 1–4, 2022. doi: 10.1109/ICEEICT53079.2022.9768622.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation, 2019. URL https://arxiv.org/abs/1811.05868.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12(null):2539–2561, November 2011. ISSN 1532-4435.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs, 2017. URL https://arxiv.org/abs/1704.02901.

Wanhua Su, Yan Yuan, and Mu Zhu. Threshold-free evaluation of medical tests for classification and prediction: Average precision versus area under the roc curve, 2013. URL https://arxiv.org/abs/1310.5103.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction, 2016. URL https://arxiv.org/abs/1606.06357.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL https://arxiv.org/abs/1710.10903.

Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014. ISSN 0001-0782. doi: 10.1145/2629489. URL https://doi.org/10.1145/2629489.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019. URL https://arxiv.org/abs/1810.00826.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings, 2016. URL https://arxiv.org/abs/1603.08861.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling, 2019. URL https://arxiv.org/abs/1806.08804.

Mu Zhu. Recall, precision and average precision. 09 2004.