Intrinsic Dimensionality and Graph Representations

Author:
Jaishree Janu
311119

Supervisors:
Dr. Maximilian Stubbemann
Dr. Dr. Lars Schmidt-Thieme

16th August 2024

Thesis submited for
Masters Thesis in Data Analytics
Master of Science in Data Analytics

Wirtschaftsinformatik und Maschinelles Lernen
Stiftung Universität Hildesheim
Universitatsplätz 1, 31141 Hildesheim

Statement as to the sole authorship of the thesis:

Intrinsic Dimensionality and Graph Representations. I hereby certify that the master's thesis named above was solely written by me and that no assistance was used other than that cited. The passages in this thesis that were taken verbatim or with the same sense as that of other works have been identified in each individual case by the citation of the source or the origin, including the secondary sources used. This also applies for drawings. sketches, illustration as well as internet sources and other collections of electronic texts or data, etc. The submitted thesis has not been previously used for the fulfilment of a degree requirements and has not been published in English or any other language. I am aware of the fact that false declarations will be treated as fraud.

16th August 2024, Hildesheim

Abstract

There has been continuous research on finding the Intrinsic Dimensionality (ID) of graphs and also on developing advanced algorithms for graph representations. We know that graph representations are exceptionally good at preserving the original structures. But what about their innate behaviors? How does innate structure of graphs change when transformed into embeddings? Do graph IDs impact the performance of downstream tasks? How do graph's various node measures correlate to graph representations? These questions are largely unexplored.

With this motivation, the thesis lays the groundwork in a new direction. This work has contributed towards synthetic graph generation for the thorough study of underlying behaviors. Further, the analysis of experiments shows that graph IDs indeed impact the evaluation metrics of various downstream tasks which are performed on the graph representations. We also have some interesting results on how distinctly graph representations obtained from different Graph Neural Networks perform on the downstream tasks.

Contents

Chapter 1

Introduction

Understanding the intrinsic (innate) structure of high-dimensional network datasets has become increasingly crucial to uncover complexity of networks. Fortunately, a significant amount of research has been done to develop algorithms which aim to find Intrinsic Dimensionality of complex networks. We also have a large number of advanced algorithms to find embeddings of such networks that preserve structure and to perform machine learning on graphs. However, to what extent does Intrinsic Dimensionality of graphs affect the ID of their Embeddings and the performance of graph applications, is yet unknown. Moreover, we also do not know how graph measures correlate to Graph Embeddings' ID.

The primary objective of the thesis is to determine if Intrinsic Dimensionality of graphs impacts Intrinsic Dimensionality of their Embeddings[1] and the performance of downstream tasks such as Node Classification, Link Prediction and Anomaly Detection.

The second objective is to find out how graph metrics such as degree centrality, betweenness centrality, and node features correlate to ID of graph Embeddings.

Background and Motivation

Intrinsic dimensionality (ID) has gained considerable attention in machine learning and data analysis. It refers to the minimum number of parameters or coordinates required to represent a dataset's underlying structure without significant information loss (Fukunaga and Olsen, 1971).

[1]Graph Embeddings refer to Node Embeddings. Edge Embeddings have not been accounted for in the experiments.

Recent advancements in graph representation learning have led to the development of various embedding techniques that aim to capture the structural properties of graphs in lower-dimensional vector spaces (Hamilton et al., 2017, Bojchevski and Günnemann, 2018, Ansuini et al., 2019, Savić et al., 2023). These embeddings serve as a bridge between the discrete world of graphs and the continuous space of vectors, enabling the application of traditional machine learning algorithms to graph-structured data. **However, the relationship between the ID (innate or natural structure) of original graphs and ID of their embeddings remains largely unexplored**.

This research is motivated by two primary factors:

1. The desire to uncover patterns between graphs' ID and corresponding embeddings' IDs.

2. To explore the relationship of graph measures with graph embeddings' ID.

By investigating these aspects, this thesis explores the correlation among the graph IDs, IDs of graph embeddings, and other graph metrics. By examining these relationships, we hope to show how graphs' complexity manifests in their representations and how this relates to established graph-theoretic measures.

This knowledge can impact various applications, including link prediction, node classification, and anomaly detection.

Research Questions

This thesis will focus on the following key research questions based on the motivations outlined above:

1. How does the intrinsic dimensionality of graphs co-relate to the intrinsic dimensionality of their embeddings? In other words, what patterns or relationships can be identified between the IDs of graphs and the IDs of their graph embeddings?

2. How do different graph embedding techniques affect the intrinsic dimensionality of the resulting representations?

3. Is there any linear correlation between the graph metrics and Graph's Node Emebeddings' IDs?

4. How do graph IDs and graph embedding IDs impact the performance of downstream tasks?

5. How can we create appropriate synthetic datasets to study the above relationships?

To answer these questions, we will employ a variety of algorithms for estimating intrinsic dimensionality and computing graph embeddings. For graph ID estimation, we will utilize state-of-the-art methods such as NC-LID (Savić et al., 2023) and GeoL (Stubbemann et al., 2023) specifically designed to handle graph structures. We will explore techniques including Node2Vec (Grover and Leskovec, 2016) and graph Neural Networks (Hamilton et al., 2017) for graph embeddings.

The ID of the resulting embeddings will be evaluated using Euclidean-based techniques, including Principal Component Analysis (PCA) (Wold et al., 1987), maximum likelihood estimation (Levina and Bickel, 2004), and fractal-based approaches (Mo and Huang, 2012). This multi-step approach aims to provide a comprehensive view of the dimensionality characteristics of both graphs and their representations.

Structure of the Thesis

This thesis is structured as follows:

- Chapter 2: Related Work

  This chapter reviews the existing literature which is closely related to our research and talks about the nuances of graph representations in the context of graph IDs.

- Chapter 3: Methodology

  This chapter details the ID estimators, the graph embedding techniques, the downstream tasks and the data collection process.

- Chapter 4: Experiments

  This chapter presents the results tables and figures from the analysis. It looks at any patterns between graph IDs and the IDs of their embeddings and analyzes the correlation between various graph metrics and graph embeddings. This chapter also interprets the results in the context of the research questions, discusses the significance of the findings, identifies the study's limitations, and suggests directions for future research.

- Chapter 5: Conclusion

  This chapter summarizes the thesis's key findings, reiterates its contributions to understanding the relationship between ID and graph representations, and provides final thoughts on the research's importance and potential impact on the field.

By following this structure, we aim to provide a thorough investigation into the ID of graphs and their representations, contributing valuable insights to graph representation learning and data analysis.

Chapter 2

Related Work

Some existing research works talk about the importance of graph representations and their IDs, these papers below discussed are closely related to our research.

The research by (Ansuini et al., 2019), investigates the ID of data representations in deep neural networks (DNNs) to understand their geometric properties and relationship with the generalization performance. It studies how the ID of data representations changes across the layers. The authors use the TwoNN estimator, a recently developed method for estimating the global ID (see section 3.1) of data representations. The TwoNN method estimates ID by analyzing the ratio of distances to the first and second nearest neighbors of each data point. Formally, given a data point x_i uniformly sampled from a manifold with intrinsic dimension d, and $r_i^{(1)}$ and $r_i^{(2)}$ as the distances to the first and second nearest neighbors of i, respectively, the ratio $\mu_i = \frac{r_i^{(2)}}{r_i^{(1)}}$ follows a Pareto distribution with parameter $d + 1$. The likelihood of the vector $\mu = (\mu_1, \mu_2, \ldots, \mu_N)$ is:

$$P(\mu|d) = d^N \prod_{i=1}^{N} \mu_i^{-(d+1)}$$

The ID d is then inferred by maximizing this likelihood or through linear regression using the empirical cumulative distribution of μ. The paper finds that in trained deep neural networks, the ID of data representations first increases and then decreases across the layers, with the ID of the last hidden layer being a strong predictor of the network's classification accuracy. This dimensionality compression is not observed in untrained networks or networks trained on randomized labels, suggesting that effective training transforms data into low-dimensional, non-linear manifolds. The study emphasizes that the ID derived from the TwoNN estimator provides a ro-

bust and computationally efficient measure that correlates with the network's ability to generalize.

The another related research (C. Li et al., 2018), presents a method to measure the ID of objective landscapes in neural networks by training them in smaller, randomly oriented subspaces instead of their full parameter space. The ID is determined by gradually increasing the subspace dimension d and noting where solutions first appear, defined as d_{int}. This approach reveals that many problems have smaller IDs than expected, indicating that neural networks often use fewer effective parameters than their total count suggests.

To measure d_{int}, the paper defines the parameter vector $\Theta^{(D)}$ in the full space as

$$\Theta^{(D)} = \Theta_0^{(D)} + P\Theta^{(d)},$$

where P is a randomly generated projection matrix, $\Theta_0^{(D)}$ is an initial parameter vector, and $\Theta^{(d)}$ is a parameter vector in the smaller subspace. This setup ensures training starts in a well-conditioned region of the parameter space. Training proceeds by optimizing $\Theta^{(d)}$ and tracking performance improvements.

Experiments on various datasets (MNIST, CIFAR-10, ImageNet) and network types (fully connected, convolutional) show consistent intrinsic dimensions across different model sizes for the same dataset. For example, a fully connected network on MNIST has $d_{\text{int}} \approx 750$. This leads to a significant network compression technique, where only the necessary dimensions are stored, reducing storage requirements drastically.

By comparing intrinsic dimensions across tasks, the paper quantifies problem difficulty, finding, for instance, that solving the inverted pendulum problem is significantly easier than classifying MNIST digits. The method also provides insights into the optimization landscape of neural networks, suggesting that large parameter spaces primarily increase the solution manifold's dimensionality rather than the intrinsic complexity of the problem itself.

The research work (Savić et al., 2023), not only proposes a local ID estimator but also presents a LID based novel embeddings algorithm called LID-elastic node2vec. It discusses the concept of Local Intrinsic Dimensionality (LID) (see section 3.1) in the context of graphs and proposes the NC-LID metric, which quantifies the discriminability of the shortest path distance considering natural communities of nodes. The authors highlight that existing distance-based LID estimators are designed for tabular datasets and are not directly applicable to graphs due to their dimensionless nature and specific properties like small-world and scale-free characteristics.

NC-LID is introduced as a measure to assess the complexity of a node's locality by substituting a ball around a data point with a subgraph around

a node. The NC-LID of a node n is defined as:

$$\text{NC-LID}(n) = -\ln\left(\frac{|S|}{T(n,S)}\right)$$

where $|S|$ is the number of nodes in the natural community S of n, and $T(n,S)$ is the number of nodes whose distance from n is less than or equal to the maximum distance between n and any node in S. The natural community of a node is determined using the fitness-based algorithm by (Lancichinetti et al., 2009), which maximizes the community fitness function considering intra-community and inter-community links.

The authors propose two LID-elastic variants of the node2vec graph embedding algorithm, in which hyperparameters are personalized per node and adjusted according to NC-LID values. The first variant, lid-n2v-rw, adjusts the number and length of random walks based on NC-LID values. The second variant, lid-n2v-rwpq, further adjusts the return and in-out parameters controlling random walk biases.

Empirical evaluation on 18 real-world graphs shows that NC-LID correlates strongly with link reconstruction errors in node2vec embeddings, outperforming traditional node centrality metrics (degree, betweenness, closeness, eigenvector centrality, and core index). The LID-elastic node2vec variants demonstrate improvements in preserving graph structure, with significant gains in F1 scores for several datasets.

Another interesting research talks about *sensitive node attributes* and their impact on node embeddings (N. Wang et al., 2022). It also proposes learning method for node embeddings from an underlying bias-free graph that isn't influenced by sensitive attributes. The authors suggest two complementary methods to uncover this underlying graph: a weighting-based method and a regularization-based method.

1. The weighting-based method reweighs the graph reconstruction loss using importance sampling on each edge, ensuring that the loss reflects a bias-free graph in expectation. Formally, this method adjusts the per-edge loss L_{edge} by the ratio $\frac{\tilde{R}(\tilde{a}_{uv})}{R(a_{uv})}$, where $\tilde{R}(\tilde{a}_{uv})$ and $R(a_{uv})$ capture the probabilities of attribute combinations in the bias-free and original graphs, respectively.

2. The regularization-based method adds a constraint that ensures the edge probabilities, conditioned on the node embeddings, remain consistent regardless of the presence of sensitive attributes. This is achieved by minimizing the discrepancy between the scores $s_\theta(z_u, z_v)$ for embedding pairs with and without sensitive attributes using an ℓ_2 regularization term.

The proposed methods suggest leading to a significant reduction in bias while maintaining competitive performance on downstream tasks, such as link prediction and node classification.

The research work by (Procházka et al., 2023) proposes a method to link graph properties with the performance of Graph Neural Networks (GNNs) on specific tasks by using a surrogate regression model. The method begins by representing the graph dataset through aggregated properties, which capture global-level information about the graph. These properties include node count, class ratio, number of components, average node degree, assortativity, attribute similarity, and various forms of homophily, among others. The surrogate model is then trained to predict the GNN performance using these graph properties.

The GNN model's performance, denoted as μ, is measured by several metrics such as log-loss, ROC-AUC, and precision-at-top-10. The surrogate model, M_{meta}, is trained to predict this performance based on the graph properties $\{P_i\}$, with optional inclusion of GNN hyper-parameters $\{H_i\}$, making the prediction $\hat{\mu} = M_{\text{meta}}(\{P_i\} \cup \{H_i\})$.

To evaluate the usefulness of the graph properties, the paper employs Shapley Additive Explanations (SHAP) to interpret the meta-model's predictions. Important properties identified include various forms of homophily, which generally show that higher homophily leads to better GNN performance.

The method also extends to hyper-parameter optimization by incorporating hyper-parameters into the surrogate model. The approach involves training the surrogate model on a dataset of graph properties and corresponding GNN performance, then using this model to predict the best hyper-parameter setup for new datasets, significantly reducing computational resources compared to traditional grid-search methods.

Experimental results on several public graph datasets demonstrate the generalization capability of the surrogate model and its effectiveness in hyper-parameter tuning, achieving performance close to optimal with a fraction of the computational effort. The findings suggest that the surrogate model can effectively use graph properties to both predict GNN performance and optimize hyper-parameters.

(Heist et al., 2023) introduces a framework to evaluate Knowledge Graphs via downstream tasks. The KGrEaT framework evaluates the utility of knowledge graphs (KGs) by their performance on downstream tasks (e.g., classification, regression, recommendation) rather than intrinsic metrics like accuracy or completeness. KGrEaT assesses how different knowledge graphs enhance task performance when used as background knowledge.

KGrEaT automates the mapping of a KG to various datasets and com-

putes performance metrics for predefined tasks. It consists of three main stages: Preprocessing, Mapping, and Task execution, each implemented as isolated Docker containers for flexibility.

1. Preprocessing Stage: Generates entity embeddings and indices for Approximate Nearest Neighbor (ANN) search. Supported embedding models include TransE, TransR, DistMult, RESCAL, and RDF2Vec.

2. Mapping Stage: Entities of the KG are mapped to dataset entities using configurable mappers. The Same-As mapper uses KG links, while the Label mapper uses token-based edit distance to match labels.

3. Task Stage: Executes different task types (e.g., classification, regression, clustering) across various datasets and algorithms. For classification, metrics like Accuracy are used, while other tasks use relevant metrics (e.g., RMSE for regression, ARI for clustering). Embeddings serve as features for models, and tasks like recommendation use distance between entity embeddings.

Experiments conducted with KGs like DBpedia, YAGO, Wikidata, and CaLiGraph show that DBpedia generally performs well across tasks. Larger KGs (YAGO, Wikidata, CaLiGraph) exhibit strong performances, particularly in recall-oriented settings. Results emphasize the need for extrinsic metrics to complement intrinsic evaluations and provide a comprehensive view of a KG's utility.

Chapter 3

Methodology

3.1 Intrinsic Dimensionality

Introduction

"An Algorithm for Finding Intrinsic Dimensionality of Data" by (Fukunaga and Olsen, 1971) mentions Intrinsic Dimensionality as the number of governing parameters. In other words, ID discovers the meaningful low-dimensional structures hidden in their high-dimensional observations. Here, "meaningful" refers to finding hidden structures that capture original information without any loss.

The concept of ID is related to dimensionality reduction and manifold learning. The ID is usually much lower than the original dimensionality of the data.

Local vs Global ID estimators Local estimators work by figuring out the dimensionality of data points based on their nearby neighbors. They assume that the data is arranged on a shape (manifold), and this shape can have different dimensions in different areas (Farahmand et al., 2007). These local estimators look at the distances or angles between nearby points, allowing them to adjust to changes in the data's structure. This kind of measurement is helpful in tasks like search, classification, and outlier detection in machine learning and data analysis because it reflects how well similarity measures can differentiate between data points. Many local ID estimators are developed and examined using methods from extreme value theory, using techniques like maximum likelihood estimation (MLE) (Levina and Bickel, 2004) and the method of moments (MoM) (Somorjai, 1986).

In contrast, global estimators provide a single dimensionality value that describes the entire dataset. They assume that the data is on or close to a manifold with a consistent dimensionality throughout. Global methods are

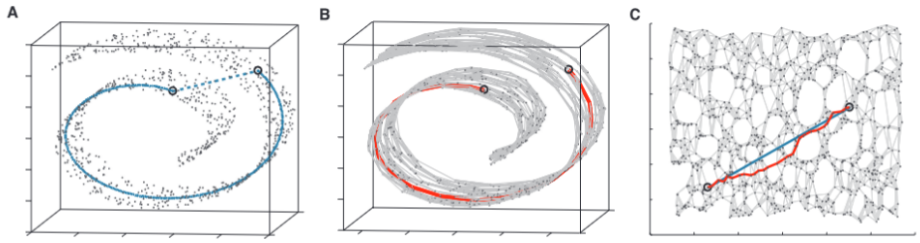

Figure 3.1: **A**: For two arbitrary points (circled) on a nonlinear manifold, their Euclidean distance in the high-dimensional input space (length of dashed line) may not accurately reflect their intrinsic similarity, as measured by geodesic distance along the low-dimensional manifold (length of solid curve). **B**: The red line distance between the two points (along the curve) is the true geodesic path between two points. **C**: The two-dimensional embedding recovered by Isomap in step three, which best preserves the shortest path distances in the neighborhood graph (overlaid) (Tenenbaum et al., 2000).

useful when you need an overall description of the data and typically analyze the whole distance matrix or the overall distribution of distances between pairs of points.

Geodesic Distances: The term "geodesic" refers to the shortest path between two points on a curved surface (Granata and Carnevale, 2016). Mathematically, a geodesic is defined as the shortest route that stays on that surface. For instance, on a flat surface (like a two-dimensional plane), the geodesic distance is simply a straight line connecting two points. However, on a curved surface such as a sphere, geodesic paths are parts of great circles, like the lines of longitude or the equator on Earth. In graph theory, the geodesic distance means the shortest path between two points (vertices) in a graph. This is measured by counting the number of edges that make up the shortest connection between them, similar to the shortest path concept on surfaces.

The concept of ID and geodesic distances is demonstrated through the classic swiss-roll example in Fig 3.1.

Methods for ID Estimation in Euclidean spaces

Correlation Dimension and fractal-based methods:
The **Correlation Dimension** is a measure used to estimate the ID of a dataset. It is defined mathematically as:

$$D_2 = \lim_{r \to 0} \frac{\log C(r)}{\log r}$$

where $C(r)$ is the correlation sum, given by:

$$C(r) = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \Theta(r - \|x_i - x_j\|)$$

Here, N is the number of data points, Θ is the Heaviside step function, and $\|x_i - x_j\|$ is the distance between points x_i and x_j. The correlation dimension can be estimated by plotting $\log C(r)$ against $\log r$ and measuring the slope of the linear part of the curve (Mo and Huang, 2012).

Fractal-based methods, such as the Box-Counting Dimension, leverage the self-similar properties of datasets to estimate their intrinsic dimensionality. These methods are particularly useful for datasets that exhibit fractal characteristics (Grassberger and Procaccia, 1983).

The Box-Counting Dimension D is defined as:

$$N(\epsilon) \propto \epsilon^{-D}$$

where $N(\epsilon)$ is the number of boxes of size ϵ needed to cover the dataset. By plotting $\log N(\epsilon)$ against $\log \epsilon$, the slope of the linear region provides an estimate of the fractal dimension.

DANCo (Dimensionality from Angle and Norm Concentration algorithm):
The DANCo algorithm is a robust intrinsic dimensionality estimator that leverages the concentration of angles and norms in high-dimensional spaces (Ceruti et al., 2012).

Let $\mathbf{x}_i$ and $\mathbf{x}_j$ be two neighboring points. The normalized distance d_{ij} is given by:

$$d_{ij} = \frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{\sigma}$$

where σ is the standard deviation of the distances.

The angle θ_{ij} between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is computed as:

$$\cos(\theta_{ij}) = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$$

The Kullback-Leibler (KL) divergence between the observed distribution P and the theoretical distribution Q is:

$$D_{KL}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

The algorithm proceeds with the following steps:

1. Compute nearest neighbors and normalized distances and angles.
2. Estimate empirical distributions of distances and angles.
3. Compute KL divergences between empirical and theoretical distributions.
4. Select dimensionality minimizing the sum of KL divergences.

(Ceruti et al., 2012) show that DANCo effectively estimates intrinsic dimensionality by leveraging angle and norm concentration, providing robustness and accuracy.

KNN:
K-Nearest Neighbors (KNN) is a versatile algorithm commonly used for classification and regression tasks. In their work, (Costa et al., 2005) adapted KNN to estimate the local intrinsic dimension of a dataset. The core idea is to determine the intrinsic dimensionality of a local neighborhood around a data point.

For a given data point x_i in a dataset X, the KNN algorithm identifies its k nearest neighbors, denoted as $N_k(x_i)$. These neighbors form a local subspace around x_i. The dimension of this subspace is assumed to approximate the local intrinsic dimension at x_i.

To estimate the local intrinsic dimension, a local covariance matrix C_i is computed for the neighborhood $N_k(x_i)$:

$$C_i = \frac{1}{k-1} \sum_{x_j \in N_k(x_i)} (x_j - \bar{x}_i)(x_j - \bar{x}_i)^T,$$

where $\bar{x}_i$ is the mean of the points in $N_k(x_i)$. The eigenvalues of C_i represent the principal components of the local data distribution. If the first m eigenvalues are significantly larger than the rest, then the estimated local intrinsic dimension is m (Costa et al., 2005).

MADA (Manifold-Adaptive Dimension Estimation algorithm):
The Manifold-Adaptive Dimension Estimation (MADA) algorithm, as detailed by (Farahmand et al., 2007), aims to estimate the ID of data lying on a low-dimensional manifold within a high-dimensional space. The algorithm leverages local neighborhood information to produce dimension estimates that are manifold-adaptive, meaning their convergence rates depend solely on the intrinsic dimension of the manifold, not on the ambient space's dimensionality.

The MADA algorithm works by first estimating the dimensionality locally around each data point using nearest-neighbor techniques. Specifically, it considers the distance to the k-nearest neighbors and uses this information to compute the local dimension. The core of the dimension estimation relies on the relationship:

$$P(X_i \in B(x,r)) = \rho(x,r)r^d$$

where $P(X_i \in B(x,r))$ is the probability of a data point X_i falling within a ball $B(x,r)$ of radius r centered at x, and $\rho(x,r)$ is a slowly varying function. The local dimension d is then estimated by fitting a line to the log-transformed relationship:

$$\ln(P(X_i \in B(x,r))) = \ln(\rho(x,r)) + d\ln(r)$$

By varying the center point x, the algorithm obtains multiple local dimension estimates, which it then combines to form a global dimension estimate. The combined global estimate can be computed either by averaging these local estimates:

$$\hat{d}_{avg} = \left\lfloor \frac{1}{n}\sum_{i=1}^{n} \hat{d}(X_i) \wedge D \right\rfloor$$

or by voting among them:

$$\hat{d}_{vote} = \arg\max_{d' \in \mathbb{N}^+} \sum_{i=1}^{n} \mathbb{I}_{\hat{d}(X_i)=d'}$$

where $\mathbb{I}$ is the indicator function. The authors proved that the convergence rate of the dimension estimates is independent of the dimensionality of the input space, thereby demonstrating the manifold adaptivity of their approach. Their experimental results support the theoretical findings, showing that the MADA algorithm performs well across various datasets, including artificial and real-world data (Farahmand et al., 2007).

MLE (Maximum Likelihood Estimation):
Maximum likelihood estimation (MLE) is a fundamental statistical method for estimating the parameters of a probability distribution, first developed by (R. A. Fisher, 1922). The principle of MLE is to choose the parameter values that maximize the likelihood function, which represents the probability of observing the given data under the assumed model. Mathematically, for a set of independent and identically distributed observations $X = (x_1, ..., x_n)$ and a parameter vector θ, the likelihood function is defined as:

$$L(\theta|X) = \prod_{i=1}^{n} p(x_i|\theta)$$

where $p(x_i|\theta)$ is the probability density function. The MLE estimate $\hat{\theta}$ is obtained by solving the optimization problem:

$$\hat{\theta} =_\theta L(\theta|X)$$

This method is widely used due to its desirable properties such as consistency, efficiency, and invariance under certain conditions.

In the context of intrinsic dimensionality estimation, MLE has been applied to develop robust estimators. For instance, (Levina and Bickel, 2004) proposed an MLE-based approach for estimating the intrinsic dimension of high-dimensional datasets. Their method models the number of points falling into a small sphere around a data point as a Poisson process, with the rate of the process depending on the intrinsic dimension. The likelihood function for a single observation x_i is given by:

$$L_i(m) = \frac{1}{k-2} \sum_{j=1}^{k-1} \log \frac{T_k(x_i)}{T_j(x_i)}$$

where m is the intrinsic dimension, k is the number of nearest neighbors considered, and $T_j(x_i)$ is the Euclidean distance from x_i to its j-th nearest neighbor. The MLE estimate for m is then obtained by maximizing the sum of these log-likelihoods over all data points. This approach demonstrates how MLE can be adapted to complex problems in data analysis, providing a statistically principled framework for dimensionality estimation that can handle the challenges of high-dimensional spaces and varying sample densities.

MOM (Method of Moments):
The method of moments is a classical statistical technique for estimating parameters of probability distributions, introduced by (Pearson, 1894). This method equates sample moments, calculated from observed data, to the theoretical moments of the distribution to estimate unknown parameters. The k-th moment of a distribution is defined as the expected value of the k-th power of the random variable:

$$\mu_k = E[X^k] = \int_{-\infty}^{\infty} x^k f(x) dx$$

where $f(x)$ is the probability density function. By matching these theoretical moments with their empirical counterparts computed from the data, one can derive equations to solve for the unknown parameters. This approach is often computationally simpler than maximum likelihood estimation, especially for complex distributions.

In the context of ID estimation, the method of moments has been applied to develop efficient estimators. For instance, (Amsaleg et al., 2015) proposed a method of moments estimator for ID based on the theory of extreme value statistics. Their approach leverages the statistical properties of nearest-neighbor distances in high-dimensional spaces to estimate the ID of datasets. Let $X_{k,n}$ be the k-th nearest neighbor distance for a query point in a dataset of size n. The authors show that under certain conditions, the random variable $-\log X_{k,n}$ follows a Gamma distribution with shape parameter k and scale parameter $1/m$, where m is the intrinsic dimension. They then use the method of moments to estimate m:

$$\hat{m} = \frac{k-1}{\frac{1}{N}\sum_{i=1}^{N} \log \frac{X_{k,n}(x_i)}{X_{k-1,n}(x_i)}}$$

where N is the number of query points used for estimation. This application demonstrates how the method of moments can be adapted to tackle modern challenges in data analysis and machine learning, providing a computationally efficient alternative to more complex estimation techniques.

MinD_ML:
The MiND_ML (Minimum Neighbor Distance - Maximum Likelihood) method, introduced by (Rozza et al., 2012), is an approach for estimating the ID of datasets. This method is based on analyzing the distribution of distances between data points in high-dimensional spaces. The key insight behind MiND_ML is that as the dimensionality of the data increases, the distribution of pairwise distances between points becomes more concentrated around a central value.

The MiND_ML algorithm operates by examining the minimum neighbor distances (MiNDs) of data points. It constructs a statistical model of how these distances should behave in spaces of different dimensionalities and then uses maximum likelihood estimation to determine the intrinsic dimension that best fits the observed data. Mathematically, for a dataset $X = \{x_1, ..., x_N\}$, the MiND_ML estimator is defined as:

$$\hat{m} =_m \sum_{i=1}^{N} \log f_m(r_i)$$

where m is the intrinsic dimension, r_i is the minimum neighbor distance for point x_i, and f_m is the probability density function of the minimum neighbor distances in an m-dimensional space. This approach is particularly effective for high-dimensional datasets and is robust in the presence of noise. The method's performance has been demonstrated on both artificial

and real-world datasets, where it has often outperformed other dimensionality estimation techniques, especially in cases of high ID.

PCA:

Principal Component Analysis, introduced by Pearson in 1901 and further developed by (Hotelling, 1933), is one of the oldest and most widely used dimensionality reduction techniques. (Wold et al., 1987) provide a comprehensive overview of PCA, explaining its mathematical foundations, applications, and interpretation. PCA works by identifying orthogonal directions of maximum variance in the data. The number of significant principal components can be used as an estimate of the ID of the dataset. Mathematically, for a dataset X with n samples and p features, PCA finds a linear transformation W such that: $Y = XW$ where Y is the transformed dataset, and the columns of W are the principal components. The ID can be estimated by examining the eigenvalues of the covariance matrix of X and selecting the number of components that explain a significant portion of the variance (Wold et al., 1987). While PCA is not specifically designed for graphs, it forms a foundation for understanding dimensionality reduction techniques and can be applied to graph embeddings to analyze their ID.

The paper *"Intrinsic dimension estimation of data by principal component analysis"* (Fan et al., 2010) proposes a PCA-based method for estimating the intrinsic dimensionality (ID) of data through the following steps:

1. Local Structure Examination: The dataset is divided into subsets, with PCA applied to each subset to examine local structures.

2. Noise Filtering: A revised PCA procedure filters out noise, focusing on significant principal components that capture essential data structures.

3. Stable Estimation: The method converges to a stable ID estimation as the neighborhood region size increases.

4. Eigenvalue Analysis: ID is estimated by analyzing the eigenvalues of the covariance matrix, with the number of significant eigenvalues indicating the intrinsic dimensionality.

5. Effectiveness: The method provides accurate and stable ID estimates, validated through experiments on synthetic and real-world datasets.

Methods for ID Estimation in non-Euclidean spaces

When it comes to non-Euclidean spaces, the idea of intrinsic dimensionality can be slightly more complex due to the different geometrical properties of these spaces compared to flat, Euclidean spaces. In non-Euclidean spaces, curvature plays a significant role in intrinsic dimensionality. Curvature dictates how the geometry of the space behaves locally and globally, which in turn affects how distances, angles, and other geometric properties are defined. For instance, in spherical geometry, there are no parallel lines, and the sum of angles in a triangle exceeds 180 degrees.

Techniques like Principal Component Analysis (PCA) may not be suitable, as they assume a Euclidean space. Instead, methods that take into account the curvature of the space are used, like multidimensional scaling for Riemannian manifolds or Isomap, which estimates geodesic distances that are more appropriate for non-Euclidean spaces (X. Wang et al., 2015).

Nearest-neighbor-based methods, tangent space and multiscale methods, graph-based methods, persistent homology dimension estimators, and projective and fractal methods each offer unique advantages and are suitable for different applications. Recent advancements have focused on improving robustness, scalability, and accuracy, particularly in non-Euclidean spaces and complex datasets.

ID estimation in complex networks and graphs

Accurate Estimation of the Intrinsic Dimension Using Graph Distances: Unraveling the Geometric Complexity of Datasets:
Granata and Carnevale, 2016 present a novel method for estimating the intrinsic dimension (ID) of datasets. The technique constructs a graph from the dataset, with nodes as data points and edges representing distances, and then calculates graph distances to approximate geodesic distances. By analyzing the probability distribution of these distances, particularly around the maximum distances, the method estimates the ID through a least-square fit to a model distribution of geodesic distances on a D-hypersphere. This approach is robust against various data complexities, such as noise and limited sampling, and does not rely on the shape of the intrinsic manifold. The paper demonstrates the method's accuracy and reliability through applications to complex datasets, including the Isomap faces database, the MNIST handwritten '2's dataset, and a protein sequence alignment dataset, showcasing its potential for broad applicability in data analysis.

Intrinsic dimension as a multi-scale summary statistics in network modeling:
Macocco et al., 2024 propose using the intrinsic dimension as a summary statistic within an Approximate Bayesian Computation (ABC) framework. This approach allows for the inference of parameters in flexible and multi-purpose mechanistic models that generate complex networks. The method can effectively compare simulated networks with real-world networks by focusing on the ID signature, ensuring that the models accurately reflect observed data.

NC-LID

Natural Community Local Intrinsic Dimensionality estimator is based on distinguishing shortest-path distance by initiating natural communities of nodes as their intrinsic localities. The length of natural communities of nodes is negatively and linearly correlated to the local-IDs of nodes. The algorithm defines graph-based local ID (GB-LID) as,

$$GBLID(n) = -\ln(\frac{|S|}{T(n,S)})$$

where S is the natural community of node n, $|S|$ is the number of nodes in S. $T(n, S)$ is the number of nodes whose distances from node n is smaller than or equal to ρ, where ρ is the maximal distance between n and any other node from S.
The natural community formed by Lancichinetti et al., 2009 algorithm is based on the concept that the nodes within a community are strongly connected compared other community nodes. The community C for a node n is computed by maximizing the fitness algorithm:

$$f_C = \frac{k_{in}(C)}{(k_{in}(C)+k_{out}(C))^{\alpha}},$$

where $k_{in}(C)$ is the total intra-degree of nodes in C, $k_{out}(C)$ is the total inter-degree of nodes in C and α is the parameter controlling the size of C.

Algorithm 1: The algorithm for computing NC-LID

Input: A graph G, a node n in G
Result: The NC-LID of n

S = Identify the natural community of node n;
M = Calculate the maximum shortest-path distance (n, z);
Q = an empty queue of nodes;
append n to Q;
mark n as visited;
// Initialize the number of nodes in $T(n, S)$ to 0;
$T = 0$;
while Q *is not empty* **do**
    c = Remove the first element from Q;
    $T = T + 1$;
    d = shortest-path-distance(n, c);
    Calculate the shortest-path distance d from n to c;
    if $d > M$ **then**
        Break;
    else
        P = Retrieve the nearest neighbors of c in G;
        for *each non-visited neighbor* p *in* P **do**
            Mark p as visited;
            append p to Q;
        end
    end
end
return $-\ln\left(\frac{|S|}{T}\right)$;

GeoL

GeoL (Stubbemann et al., 2023) is based on the approach proposed by Pestov, 2007 who established link between ID and mathematical concentration of measure phenomenon. However, the method is infeasible for large-scale datasets. GeoL proposes a computationally feasible method which incorporates neighbourhood information and geometrical properties. The calculation of ID $\delta(D) := \frac{1}{\Delta(D)^2}$ follows the following four axioms:

1. **Axiom of concentration:** A sequence of geometric datasets converges against the constant dataset, if and only if their IDs diverge against infinity.

2. **Axiom of feature antitonicity:** If dataset D_1 has more feature functions then D_0 (i.e. having potentially more information to separate data points), it should have a lower intrinsic dimension.

3. **Axiom of continuity:** If a sequence of geometric datasets converge against a specific geometric dataset, the sequence of the IDs should converge against the ID of the limit geometric dataset.

4. **Axiom of geometric order of divergence:** If a sequence of geometric datasets converges against the constant dataset, its IDs should diverge against infinity with the same order as $\frac{1}{(\Delta}$)^2 does.

Algorithm 2: The pseudo code to compute $\Delta_{s,-}(D)$, $\Delta_{s,+}(D)$, $\Delta(D)$ for a finite GD $D = (X, \mu, F)$

Result: $\Delta_{s,-}(D)$, $\Delta_{s,+}(D)$, $\Delta(D)$
Input: Finite GD $D = (X, \mu, F)$, support sequence $s = (2 = s_1, ..., s_l = |X|)$, *exact* (Boolean)

for *all* f *in* F **do**
  Compute feature sequence $l_{f,D}$;
end
$\Delta(D) = 0$;
$s_0 = 1$;
$\phi_{s_0}(D) = 0$;
for *all* i *in* $\{1, ..., l\}$ **do**
  for *all* f *in* F **do**
    $\phi_{s_i,f}(D) = \min_{j \in \{0,...,|X|-si\}} l^{f,D}_{si+j} - l^{f,D}_{1+j}$
  end
  $\phi_{s_i}(D) = \max\{\phi_{s_i,f}(D) | f \in F\}$;
  $F_{s_{i-1}} = \{f \in F | \phi_{s_i,f}(D) > \phi_{s_{i-1}}(D)\}$;
  $\Delta(D) += \phi_{s_i}(D)$;
end
$\Delta_{s,-}(D) = \frac{1}{|X|}(\Delta(D) + \sum_{i=1}^{l-1} \sum_{s_i \le j < s_{i+1}} \phi_{s_i}(D))$;
$\Delta_{s,+}(D) = \frac{1}{|X|}(\Delta(D) + \sum_{i=1}^{l-1} \sum_{s_i \le j < s_{i+1}} \phi_{s_{i+1}}(D))$;
; `// Approximation finished. Continue with exact computation if desired.`
if *exact* **then**
  for *all* i *in* $\{1, ..., l\}$ **do**
    for *all* $s_i \le j < s_{i+1}$ **do**
      for *all* f *in* F_{s_i} **do**
        $\phi_{s_i,f}(D) = \min_{j \in \{0,...,|X|-si\}} l^{f,D}_{si+j} - l^{f,D}_{1+j}$
      end
    end
    $\phi_j(D) = \max\{\phi_{j,f}(D) | f \in F_i\} \cup \{\phi_{si}\}$;
    $\Delta(D) += \phi_j(D)$;
  end
  $\Delta(D) = \frac{1}{|X|}\Delta(D)$;
  return $\Delta_{s,-}(D)$, $\Delta_{s,+}(D)$, $\Delta(D)$;
end
return $\Delta_{s,-}(D)$, $\Delta_{s,+}(D)$, $\Delta(D)$;

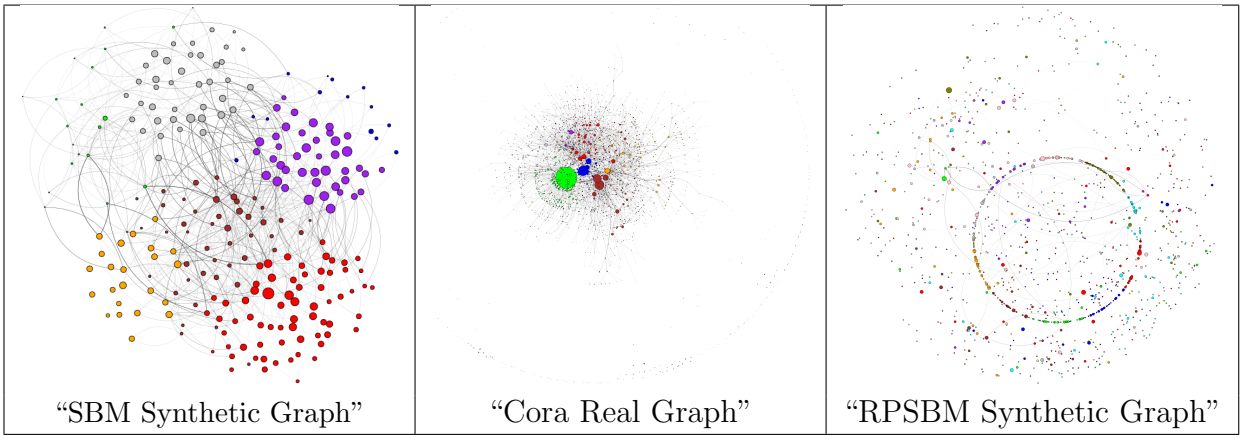

Figure 3.2: Illustration: Sample graphs used in the research

3.2 Graphs and Graph Metrics

Graphs are networks or structures which consist of entities (also called nodes or vertices) connected to other entities via relationships (also called edges or links). By their nature, graphs are complex structures that capture relationships between entities.

Number of nodes and number of edges

The number of entities or vertices in the graph is called the number of nodes. The colorful circles depicted in the figure 3.2 represent nodes. The node colors represent node labels or node classes. The number of relationships or links connecting nodes is the number of edges. The gray color interactions between nodes represent edges as depicted in the figure 3.2.

Load Centrality

The load centrality of a node is the fraction of all shortest paths that pass through that node. The load centrality of a graph is computed as the average of load centralities over all nodes (Goh et al., 2001).

Betweenness Centrality

Betweenness centrality of a node is the sum of the fraction of all-pairs shortest paths that pass through v (Brandes, 2001):

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where V is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t) paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v.

Feature Name / Graph Name	**SBM synthetic graph**	**CORA real graph**	**RPSBM synthetic graph**
Num nodes	252	2708	3000
Num edges	3616	10556	1516
Num node features	7	1433	5
Num classes	7	7	10
Degree Centrality	0.114	NA	$3.3 * 10^-4$
Closeness Centrality	0.353	NA	$2 * 10^-4$
Betweenness Centrality	0.007	NA	$1.08 * 10^-7$
Load Centrality	0.007	NA	$1.08 * 10^-7$
Num nodes per class	NA	NA	300
Node Homophily Ratio	NA	NA	$4.4 * 10^-4$
Average Degree	NA	7.79	0.5

Table 3.1: Comparison of Graph Features of graphs depicted in the Figure 3.2

Closeness Centrality

Closeness centrality of a node u is the reciprocal of the average shortest path distance to u over all $n-1$ reachable nodes (Freeman et al., 2002).

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$$

where $d(v,u)$ is the shortest-path distance between v and u, and $n-1$ is the number of nodes reachable from u. Notice that the closeness distance function computes the incoming distance to u for directed graphs. To use outward distance, act on $G.reverse()$. Notice that higher values of closeness indicate higher centrality.

Degree Centrality

The degree centrality for a node v is the fraction of nodes it is connected to. The degree centrality values are normalized by dividing by the maximum possible degree in a simple graph $n-1$ where n is the number of nodes in G.

Node Homophily Ratio

It measures the extent to which similar nodes are connected to each other in a graph. Homophily is often summarized by the adage "birds of a feather flock together," indicating that similar individuals (or nodes) are more likely to be connected.

Number of blocks

The number of blocks is the number of clusters in a graph.

Node features

The number of node features. The features used to encode the information of nodes. The node features are represented as vectors, used as input to Graph Neural Networks and crucial for various graph Machine Learning applications such as node classification, link prediction and anomaly detection.

The table 3.1 lists some graph measures for sample graphs used in the research.

3.3 Graph Embeddings, techniques and applications

Graph Embeddings

Graph representation learning methods aim to transform graph data into low-dimensional vector representations while preserving original structures. These methods can be categorized into several types (F. Chen et al., 2020):

1. **Dimensionality Reduction**: Classical methods like PCA, LDA, MDS, Isomap, and LLE reduce high-dimensional data to lower dimensions. They are mathematically transparent but struggle with capturing higher-order proximities.

2. **Random Walk-Based Methods**: Methods like DeepWalk and node2vec generate sequences from random walks on graphs to capture local and global structures. DeepWalk maximizes the probability of vertex context pairs while node2vec balances breadth-first and depth-first searches to enhance local and global structure preservation.

3. **Matrix Factorization**: Techniques like Graph Laplacian Eigenmaps and GraRep factorize adjacency or proximity matrices to preserve graph structure. These methods can capture global structures but face scalability issues.

4. **Neural Networks**: These include Graph Convolutional Networks (GCN), which aggregate neighbor embeddings, and Variational Graph Autoencoders (VGAE), which use GCNs as encoders. While state-of-the-art in performance, they rely heavily on GPU and are less interpretable.

5. **Large Graph Embedding**: Methods like LGCL and GPNN use subgraph sampling and partitioning to handle large graphs. They are efficient but still computationally intensive.

6. **Hyper-Graph Embedding**: These methods, like HGNN and DHNE, model complex relationships where edges connect multiple nodes. They provide a more nuanced representation but are harder to implement.

7. **Attention Mechanisms**: Graph Attention Networks (GAT) and AttentionWalks use attention mechanisms to focus on relevant parts of the graph, improving long-distance node modeling.

8. **Others**: Methods like GraphGAN use adversarial learning, while GenVector employs Bayesian embeddings for large social knowledge graphs.

Each method is shown to have strengths and weaknesses (F. Chen et al., 2020). Classical methods are easy to implement but limited in scope. Random-walk methods capture contextual information but may miss global structures. Matrix factorization and neural network models provide rich

representations but are computationally expensive. Large graph methods scale well but require significant resources, while hyper-graph models and attention mechanisms offer advanced capabilities with increased complexity.

Traditional Graph Representation Methods

Early approaches to graph representation focused on matrix factorization techniques and random walk-based methods. The following models mentioned are based on unsupervised learning.

DeepWalk

Perozzi et al., 2014 introduced DeepWalk, a method inspired by word2vec from natural language processing. DeepWalk learns node embeddings by performing random walks on graphs. The algorithm works as follows:

1. Perform random walks on the graph to generate sequences of nodes.
2. Treat these sequences as "sentences" and nodes as "words."
3. Apply the Skip-gram model to learn vector representations of nodes.

The objective function for DeepWalk is:

$$\min_{\Phi} -\log \Pr(\{v_{i-w}, \ldots, v_{i+w}\} \setminus \{v_i\} \mid \Phi(v_i))$$

where Φ is the matrix of node embeddings, and v_i is the current node in a random walk.

DeepWalk demonstrated significant improvements in multi-label classification tasks on social networks, showing its ability to capture meaningful structural information in the learned embeddings.

Node2Vec

Grover and Leskovec, 2016 introduced node2vec, an extension of DeepWalk that employs a flexible notion of a node's network neighborhood. Node2vec uses a biased random walk procedure to efficiently explore diverse neighborhoods, allowing it to capture the network's local and global structural properties. The key innovation of node2vec is the introduction of parameters p and q that control the random walk:

1. Return parameter p: Controls the likelihood of immediately revisiting a node.

2. In-out parameter q: Controls the tendency to explore "outward" (BFS-like) vs. "inward" (DFS-like).

Authors attempt to optimize the following objective function,

$$\max \sum_{u \epsilon v} \log Pr(N_s(u)|f(u)$$

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Here, $d_t x$ is the shortest path distance between nodes t and x. Node2vec has shown superior performance in multi-label classification and link prediction tasks across various real-world networks, suggesting that it captures important structural information about the graphs (Grover and Leskovec, 2016).

Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) embed real-world graphs into low-dimensional spaces for various tasks such as node classification, graph classification, link prediction, and clustering. GNNs have evolved into a powerful framework for managing irregular data like graphs and manifolds. They learn task-specific node/edge/graph representations through hierarchical iterative operators, allowing traditional machine-learning methods to be applied to graph-related tasks. Despite their success, GNNs face challenges like high computational costs for large graphs, sensitivity to graph structure perturbations, limitations imposed by the Weisfeiler-Leman (WL) graph isomorphism test, and the black-box nature of their mechanisms. This survey provides a comprehensive review of GNNs, presenting a novel taxonomy and summarizing four future research directions to address these challenges (Zhou et al., 2022). A simple graph $G = (V, E)$ consists of nodes V and edges E. The adjacency matrix A_G encodes the graph's structure.

Graph Convolutional Networks (GCNs)

(Kipf and Welling, 2016a) introduced Graph Convolutional Networks, which generalize convolutional neural networks to graph-structured data. GCNs learn node representations through neighborhood aggregation. A simple picture of the input and output layers is shown in Fig 3.4.

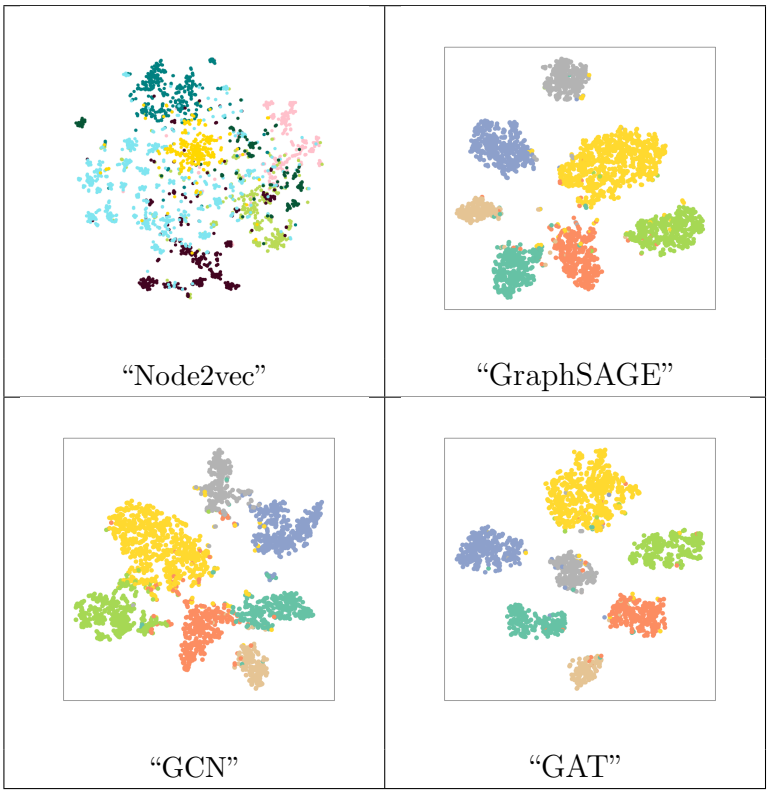

Figure 3.3: Illustration: Node embeddings for the cora_ML graph from Node2vec, GraphSAGE, GCN and GAT methods.

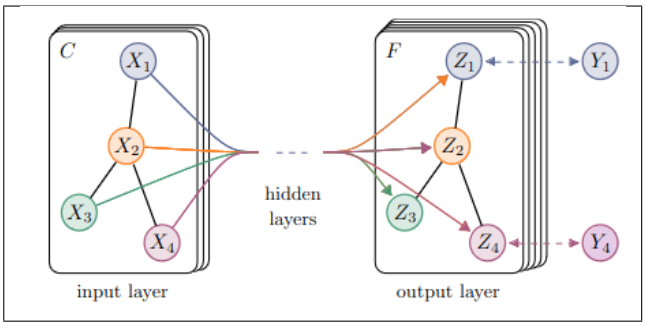

Figure 3.4: Depiction of multi-layer Graph Convolutional Network (GCN) for semi-supervised learning with C input channels and F feature maps in the output layer. The graph structure (edges shown as black lines) is shared over layers, labels are denoted by Y_i, (Kipf and Welling, 2016a).

The layer-wise propagation rule in a GCN is given by:

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

where $\tilde{A} = A + I$ is the adjacency matrix with self-loops, $\tilde{D}$ is the degree matrix of $\tilde{A}$, $H^(l)$ is the matrix of activations in the l-th layer, $W^(l)$ is the weight matrix for layer l, and σ is a non-linear activation function. GCNs have shown impressive performance on various graph-based tasks, including node classification and link prediction. This suggests that GCNs are effective in capturing important structural information and node features in graph data.

GraphSAGE

(Hamilton et al., 2017) introduced GraphSAGE, an inductive framework for learning node embeddings that can generate embeddings for previously unseen nodes. GraphSAGE employs a method that creates embeddings by extracting and combining features from the immediate vicinity of a node within the graph structure. The GraphSAGE algorithm (Fig 3.5) works as follows:

1. Sample a fixed-size set of neighbors for each node.
2. Aggregate the features of the sampled neighbors using a learnable aggregation function.
3. Combine the aggregated neighborhood features with the node's own features.

The general form of the GraphSAGE update rule is:

$$h_v^k = \sigma\left(W^k \cdot \text{CONCAT}\left(h_v^{(k-1)}, \text{AGG}_k\left(\{h_u^{(k-1)}, \forall u \in N(v)\}\right)\right)\right)$$

where h_v^k is the hidden state of node v at layer $k, N(v)$ is the neighborhood of v, AGG_k is the aggregation function at layer k, and W^k is a learnable weight matrix.

GraphSAGE has shown effectiveness across various graph structures and tasks, including node classification and link prediction (Hamilton et al., 2017).

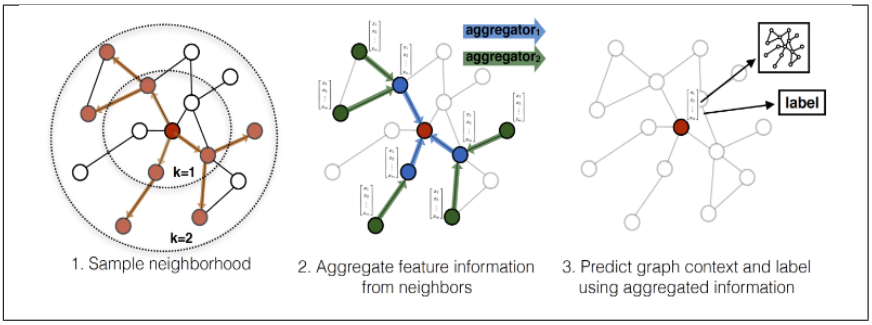

Figure 3.5: Illustration of GraphSAGE sample and aggregate approach. (Hamilton et al., 2017)

Graph Attention Networks (GATs)

(Veličković et al., 2017) introduced Graph Attention Networks, which incorporate attention mechanisms into GNNs. GATs allow the model to assign different importance to different neighboring nodes during the aggregation process. The attention mechanism in GATs is defined as:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$

where

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

is the attention coefficient between nodes i and j, a is a learnable attention function, and W is a weight matrix. The output features for each node are then computed as:

$$\vec{h}'_i = \sigma\left(\sum_{j \in N_i} \alpha_{ij} \mathbf{W}\vec{h}_j\right)$$

GATs have demonstrated state-of-the-art performance on various graph-based tasks, showing their ability to capture complex relationships between nodes. This highlights the effectiveness of the attention mechanism in improving the model's performance on tasks such as node classification and link prediction (Veličković et al., 2017). The attention mechanism and multihead attention is shown in Fig 3.6.

Graph representations of cora_ML graph shown in fig 3.3 by four different algorithms.

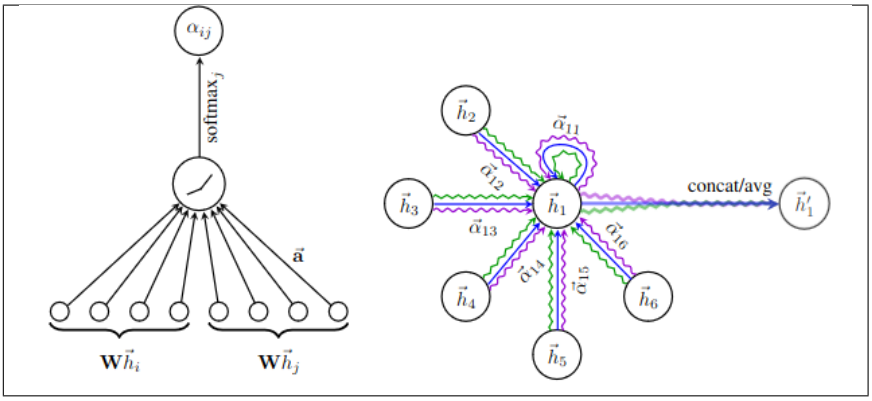

Figure 3.6: **Left**: The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by GAT, parametrized by a weight vector $\vec{a}$, applying a LeakyReLU activation. **Right**: An illustration of multihead attention (with K = 3 heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_i$. (Veličković et al., 2017)

Machine Learning applications in Graphs

The main goal of graph embedding is to represent each node's properties in a smaller vector. This makes it easier to measure node similarities using standard metrics (Xu, 2021). These embeddings can then be used for various downstream tasks like node classification (Xiao et al., 2022), link prediction (Zhang et al., 2020), community detection (Z. Chen et al., 2017), graph classification (T. Chen et al., 2019), outlier detection (Lamsal, 2024) and many more.

Node Classification

Node classification entails predicting the labels of nodes within a graph. By utilizing graph embeddings, the structural and feature information of nodes is encapsulated in a compact form, which can then be input into a classifier to predict node labels.

Three Graph Neural Networks are employed for node classification tasks: GraphSAGE (Hamilton et al., 2017), Graph Convolution Network (Kipf and Welling, 2016a), and Graph Attention Network (Veličković et al., 2017, Brody et al., 2021). The *torch_geometric* library's SAGEConv class is used to implement the GraphSAGE algorithm, GATv2Conv class for the GAT Network, and GCNConv class to implement the GCN Network[1].

[1]https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html

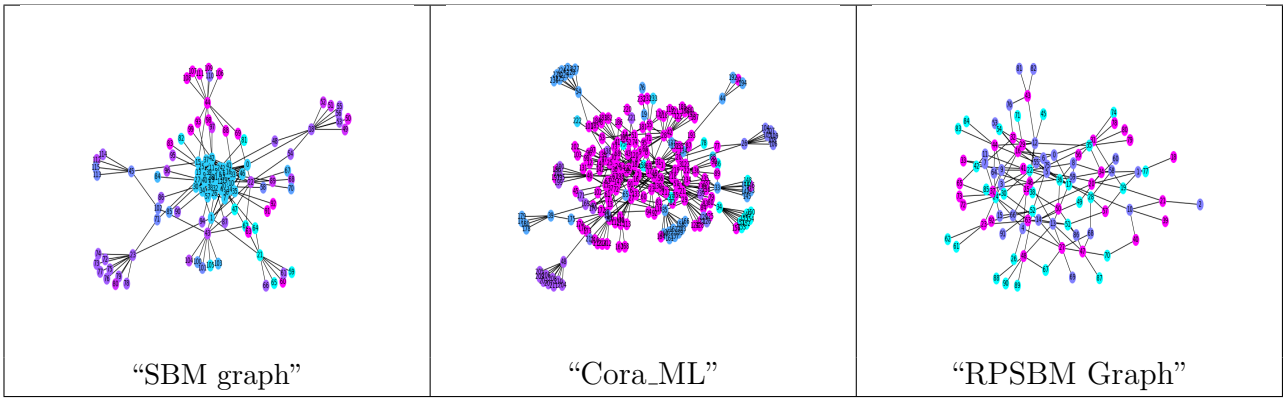

Figure 3.7: Illustration: Subgraphs generated by Neighbor Sampler, four subgraphs created for each large graph, nodes of subgraphs depicted in different colors.

Model Preparation for training

The primary task for preparation before model training is creating train, validation and test masks. We have used 80% nodes for training, 10% for validation and 10% for testing. Those masks are used for training loss calculation and model evaluation. The second crucial step is to create mini-batches. The concept of Neighbor Sampler is employed to create mini-batches in large-scale graphs where full-batch training is not feasible (Hamilton et al., 2017). $num\_neighbors$ denotes the number of sampled neighbors for every node in each iteration. For instance, [2, 7] means we want 2 neighbors and 7 of their neighbors. Fig 3.7 shows subgraphs in different colors for SBM, RPSBM and a real graph.

An illustration of node classification on a SBM graph shown in fig 3.8. Accuracy metric (see 6.2) is used to evaluate the node classifiers.
Hyperparameters for the three neural networks are listed in the table 3.2.

Link Prediction

Link prediction is a process that attempts to forecast the presence of connections between nodes in a graph. This technique is crucial for various applications, including:

- Expanding social networks
- Developing recommendation systems

	Graph SAGE (Node Classification)	**GCN (Node Classification)**	**GAT (Node Classification)**	**GAE (Link prediction)**
num neighbors	[5, 10]	[5, 10]	[5, 10]	-
batch_size	16	16	16	-
Hidden Feature Size	64	64	64	128
Depth of Network	2	2	2	2
Dropout rate	0.5	0.5	0.6	-
Loss function	Cross Entropy	Cross Entropy	Cross Entropy	BCEWith LogitsLoss
Activation Function	ReLU	ReLU	ReLU	ReLU
Learning Rate	0.01	0.01	0.005	0.01
L2 Regularization	5.00E-04	5.00E-04	5.00E-04	-
Back prop Optimizer	Adam	Adam	Adam	Adam

Table 3.2: Comparison of hyperparameters of different Graph Neural Networks for node classification and link prediction tasks.

- Completing knowledge graphs

Graph embeddings play a significant role in this process by creating compact representations of nodes. These representations facilitate the prediction of potential links based on the closeness of node embeddings in vector space.

Link prediction presents more challenges than node classification, as it requires modifications to predict edges using node embeddings. The prediction process involves the following steps:

1. Data Splitting: Utilize a transformation, such as the RandomLinkSplit[2] class, to divide edges into training, validation, and test sets.

[2]RandomLinkSplit

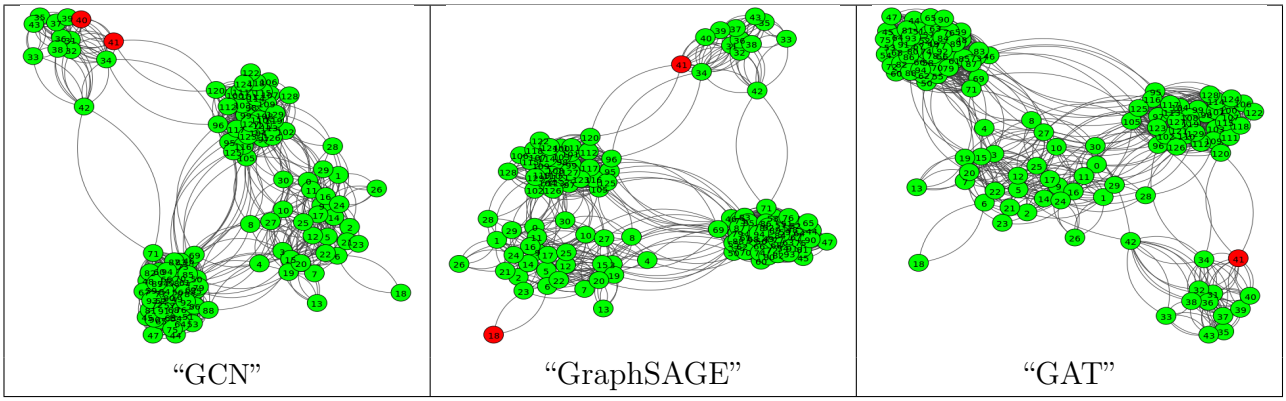

Figure 3.8: Illustration: Node Classification for a SBM graph by GCN, GraphSAGE and GAT classifiers. Green color nodes are correctly classified, red color nodes are incorrectly classified.

2. Node Embedding Creation: Implement an encoder that processes the graph using two convolution layers. The GCNConv[3] from torch_geometric is employed for this purpose.

3. Negative Link Addition: Randomly incorporate negative links into the original graph. This transforms the task into binary classification, with positive links from original edges and negative links from added edges. The torch_geometric library provides a negative_sampling()[4] function for this purpose.

4. Link Prediction: Employ a decoder to perform binary classification on all edges, including negative links, using node embeddings.

The model structure and insights are based on (Kipf and Welling, 2016b). ROC-AUC curve (see 6.2) is used to evaluate link prediction model.

Anomaly Detection

Anomaly detection in graphs involves identifying unusual patterns or outliers within graph-structured data. These anomalies can manifest as nodes, edges, or subgraphs that significantly deviate from the expected structure or behavior of the graph (Lamsal, 2024).

According to the literature, there are two primary types of outliers in graph datasets:

[3] GCNConv

[4] negative_sampling

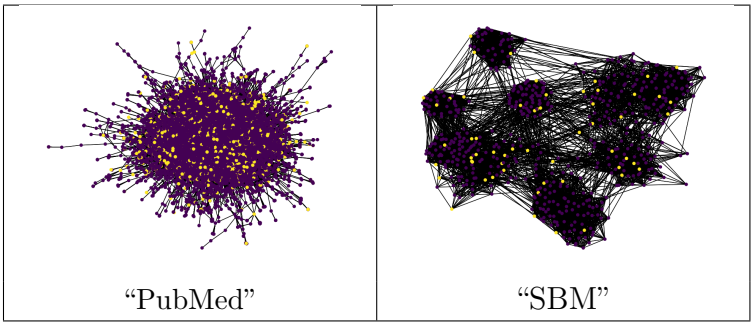

Figure 3.9: Illustration: Outliers generated using PyGOD library. Purple color nodes are normal while yellow color nodes are the structural outliers.

- **Structural Outliers**: Nodes with dense connections in contrast to sparsely connected regular nodes.
- **Contextual Outliers**: Nodes whose attributes differ significantly from their neighboring nodes (K. Liu et al., 2022).

For this anomaly detection task, we utilize the PyGOD[5] library, which is built on top of PyG (PyTorch Geometric). To create a dataset with outliers:

- We generate 10% of outlier nodes using the `gen_contextual_outlier()` function from the PyGOD module. Fig 3.9 shows outliers in SBM and real graphs.
- The outlier-injected dataset can be loaded via the PyGOD module.

To detect these outliers, we employ the DOMINANT (Deep Anomaly Detection on Attributed Networks) model (Ding et al., n.d.). This auto-encoder network uses graph convolution layers, and its reconstruction errors serve as node anomaly scores. The model has following four main components:

1. Attributed Network Encoder
2. Structure Reconstruction
3. Attribute Reconstruction
4. Error Combination

[5]PyGOD

This approach is shown to allow the detection of both structural and contextual anomalies in graph-structured data, providing a comprehensive method for identifying unusual patterns or outliers within complex network structures. The ROC-AUC curve and average precision scores 6.2 are used to evaluate the anomaly detection model.

3.4 Data Foundation

Synthetic Graph Generation

A large number of synthetic graphs with diversified features are required to extensively answer research questions. Hence, synthetic graphs with a range of parameters are generated to cover a wide range of possibilities. *torch-geometric.datasets* library is well equipped to generate such synthetic graphs. The two particular classes used in this research to generate graphs are **StochasticBlockModelDataset** (SBM) and **RandomPartitionGraph-Dataset** (RPSM) from *torch-geometric.datasets*. All the synthetic graphs generated are **undirected** and **unweighted**.

I also experimented with synthetic graphs generated from the torch's *FakeDataset* class and various NetworkX' random graph functions. I explain these graph generation methods and challenges faced with these synthetic graphs in the upcoming subsections.

StochasticBlockModelDataset

A synthetic graph dataset generated by the stochastic block model, (Anderson et al., 1992). The node features of each block are sampled from normal distributions where the centers of clusters are vertices of a hypercube, as computed by the classification method mentioned in, (Guyon et al., 2004).

A total number of 195 SBM graphs are generated using these parameter values.

RandomPartitionGraphDataset

The random partition graph dataset is inspired from (Kim and Oh, 2021). The class SBM used as the base class in torch's implementation of RPSBM. This is a synthetic graph of communities controlled by the node homophily and the average degree, and each community is considered as a class. The node features are sampled from normal distributions where the centers of clusters are vertices of a hypercube, as computed by the classification method (Guyon et al., 2004).

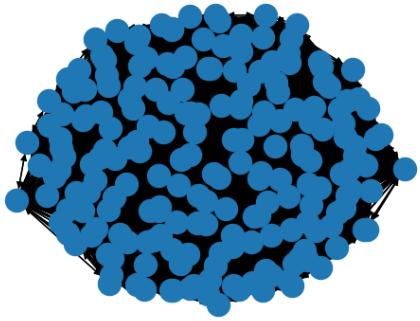

Figure 3.10: A FakeDataset sample graph, depicted to demonstrate that the graphs generated formed one big community or cluster.

The fundamental difference between SBM and RPSBM graphs is that the nodes belonging to a cluster/block in SBM graphs also belong to the same class, while nodes belonging to the same class in RPSBM graphs may belong to different clusters/communities.

A total number of 252 RPSBM graphs are generated using these parameter values.

The sample graphs generated from SBM and RPSBM are shown in fig 3.2.

FakeDataset

This is another class from torch-geometric.datasets library using which synthetic graphs were generated. The challenge associated with this class in context of the current research goals, is that it doesn't provide any parameters to create clusters with various edge densities.
A number of graphs were generated by varying its parameters such as avg_degree, avg_num_nodes and num_channels. All the graphs generated were strongly connected to each other forming only one big community. This prevented the research being carried out on a variety of graph parameters. A sample graph is shown in the figure 3.10. The graph shown was generated with inputs as: avg_num_nodes=200, avg_degree=99.0, num_channels=256 and num_classes=20.

Networkx' random graph generators

There are a large number of random graph generators from the NetworkX (Hagberg et al., 2008) library also available.
Some of the popular types of graphs generated using NetworkX functions are:

- erdos_renyi_graph(n, p): takes as input number of nodes and probability for edge creation, returns a G_n,p random graph, also known as an Erdős-Rényi graph or a binomial graph. (Bollobás, 1998)
- newman_watts_strogatz_graph(n,k,p): takes as input number of nodes, number of k-neighbors in a ring topology for each node and the probability of adding a new edge, returns Newman–Watts–Strogatz small-world graph. (Newman and Watts, 1999)
- barabasi_albert_graph(n,m): takes as input number of nodes and number of edges, returns a random graph using Barabási–Albert preferential attachment. A graph of n nodes is grown by attaching new nodes each with m edges that are preferentially attached to existing nodes with high degree. (Barabási and Albert, 1999)
- powerlaw_cluster_graph(n,m,p): takes as input number of nodes, number of edges and probability of adding a triangle after adding a random edge, returns graph with powerlaw degree distribution and approximate average clustering. (Holme and Kim, 2002)

However, experiments were not performed on Torch's FakeDataset graphs and NetworkX's random graphs. The reason being that they only generated graphs with one big community, meaning all the nodes were strongly connected to each other 3.10. It specially puts the NC-LID algorithm at a disadvantage. Note that NC-LID first forms a community for each node and then computes its Local-ID.

Real Graphs

The real graphs used in this research are from Planetoid (Cora, CiteSeer and PubMed) and CitationFull (Cora, Cora ML, DBLP, CiteSeer and Pubmed) classes of *torch-geometric.datasets*. Each of the datasets mentioned consist of a single graph.

Citation Networks

The CitationFull datasets are derived from the paper, (Bojchevski and Günnemann, 2018) and include Cora, PubMed, CiteSeer, DBLP, and Cora-ML. These datasets represent citation networks where nodes are documents (scientific publications) and edges are citation links between these documents. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary.

	Cora	Cora ML	CiteSeer	DBLP	PubMed	Cora	CiteSeer	PubMed
# nodes	19,793	2,995	4,230	17,716	19,717	2,708	3,327	19,717
# links	126,842	16,316	10,674	105,734	88,648	10,556	9,104	88,648
# classes	70	7	6	4	3	7	6	3
# features	8,710	2,879	602	1,639	500	1,433	3,703	500

Table 3.3: Real Graphs' parameters

Planetoid Networks

The citation network datasets "Cora", "CiteSeer" and "PubMed" are derived from the (Yang et al., 2016) paper. Nodes represent documents and edges represent citation links. Training, validation and test splits are given by binary masks. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. For instance, Cora has a dictionary which consists of 1,433 unique words. The Cora dataset consists of 140 training data, 500 validation data, and 1,000 test data.

Chapter 4

Experiments

4.1 Tool and libraries

Before discussing data foundation and experiments, we list all the tools and libraries that we have used in the experiments.

Deep Learning and Machine Learning in graphs

Torch is a unified library that consists of implementations of state-of-the-art machine learning algorithms, (Collobert et al., 2002). PyTorch is a machine learning library based on Torch, and it is focused on usability and speed of utilizing state-of-the-art implementations of Deep Neural Networks, (Paszke et al., 2019).
We have used the PyTorch Geometric library to apply deep learning methods on graph datasets, (Fey and Lenssen, 2019). PyTorch Geometric is also well equipped to efficiently utilize GPU and CUDA kernels. It has introduced efficient mini-batch handling for large datasets. It contains important utilities required for executing Deep Learning methods on graph datasets.
torch_geometric.nn module contains implementations of all state-of-the-art Graph Neural Networks. We have particularly used SAGEConv, GATv2Conv, and GCNConv classes from this module. PyTorch Geometric also contains a wide variety of graph datasets that can be loaded directly into torch.Data format.
We have also utilized PyGOD - a python library for Graph Outlier Detection based on PyTorch Geometric and PyTorch (link listed below). The important Pytorch-geometric links which will be referred to in the upcoming sessions are listed below:

1. Synthetic Datasets

2. StochasticBlockModelDataset
3. RandomPartitionGraphDataset
4. FakeDataset
5. Planetoid
6. CitationFull
7. GNNs
8. Anomaly Detection Library

Graphs, visualizations and analysis tools

The most popular library in the Python world for graph generation, graph analysis, and exploration is NetworkX, (Hagberg et al., 2008). Some important NetworkX links that we will refer to, in the upcoming sessions are listed below:

1. Random Graph Generators
2. Centrality graph measures
3. Networkx graph visualizations

The other tools for graph visualizations, data analysis and plots used are igraph (Csardi, 2013), matplotlib (Hunter, 2007), seaborn (Waskom, 2021), pandas (McKinney et al., 2011) and numpy (Oliphant et al., 2006).

ID estimators for Embeddings

scikit-dimension is utilised for computing IDs for Graph embeddings. A variety of ID estimators from the library are used such as DanCo, MoM, Mind_ML, PCA, etc.

Listing github repos

The implementation of ID4GeoL (Stubbemann et al., 2023) algorithm is publicly available on github[1]. The implementation of the NC-LID algorithm (Savić et al., 2023) is also openly available on github[2].

[1]https://github.com/mstubbemann/ID4GeoL
[2]https://github.com/graphsinspace/graspe

The implementation of this research can be found on the University's gitlab[3]. Please follow the experimental setup steps listed on the README.md page.

4.2 Compute Resources

University compute resources

We have run our experiments using the computational resources available at the university cluster[4]. These include high-performance computing nodes with multiple GPUs and large memory capacities, which enable efficient processing of large-scale datasets.
ISMLL (Information Systems & Machine Learning Lab) has a cluster with FAI (Fully Automatic Installation) consisting of 61 CPU nodes making up 1288 cores, 2176 GB RAM and 183 TB disk space. It uses Slurm as a batch queuing system that schedules compute jobs and the corresponding requested resources. Compute jobs of Java, C/C++, Python, R, and Matlab can be executed on the cluster.
For this research's experiments, STUD partition was used which has STUD-000 nodes, 8 CPU/GPU and 25 MEM/GPU.

4.3 Experiment Results

We have studied the relationships among our quantities of interest (graph ID, embedding ID, graph measures, and evaluation metrics of downstream tasks) by analyzing the statistical correlation of our results. We have conducted correlation analysis using Pearson co-efficient (refer 6.1), and are showing the p-value (refer 6.1) of the Student's t-test to determine the statistical significance of a correlation. The graph level granularity indicates that each graph contributes an observation point in the analysis. On the other hand, the node level granularity indicates that each node contributes an observation point in the analysis. A detailed list of the studied pair-wise correlations is available in the Appendix section 6.3. We have used a threshold of p-value < 0.05 to determine statistical significance. We have also used kernel density estimates (Gaussian kernel) to study & visualize the distributions of IDs over graph IDs and embedding IDs.

Please refer 6.2 for the definitions of evaluation metrics of Machine Learning applications on graphs : node classification, link prediction and anomaly

[3]https://www.uni-hildesheim.de/gitlab/janu/grarepid
[4]https://www.uni-hildesheim.de/gitlab/ismll/cluster-tutorial/-/wikis/home

detection.

We have used the following variable names in our tables and figures:

1. avg_nclid_graph: average of local ID (float value) of each node in a graph. (NC-LID algorithm)
2. dim_graph_geol: ID (float value) of a graph estimated by GEOL.
3. avg_nclen: average of length of Natural Community (integer value) of each node in a graph. (NC-LID algorithm)
4. sage_test_acc or sage_acc: Test accuracy of node classification of GraphSAGE algorithm. (see 6.2)
5. gcn_test_acc or gcn_acc: Test accuracy of node classification of GCN algorithm. (see 6.2)
6. gat_test_acc or gat_acc: Test accuracy of node classification of GAT algorithm. (see 6.2)
7. avg_precision_score: Average precision score metric of anomaly detection task. (see 6.2)
8. roc_auc: ROC curve (AUC) of anomaly detection task. (see 6.2)
9. degree_cent: Degree Centrality
10. close_cent: Closeness Centrality
11. between_cent: Betweenness Centrality
12. test_auc: ROC curve (AUC) of Link Prediction task. (see 6.2)
13. gcn: ID Embeddings, embeddings obtained from GCN
14. gat: ID Embeddings, embeddings obtained from GAT
15. sage: ID Embeddings, embeddings obtained from GraphSAGE

Synthetic graphs: SBM

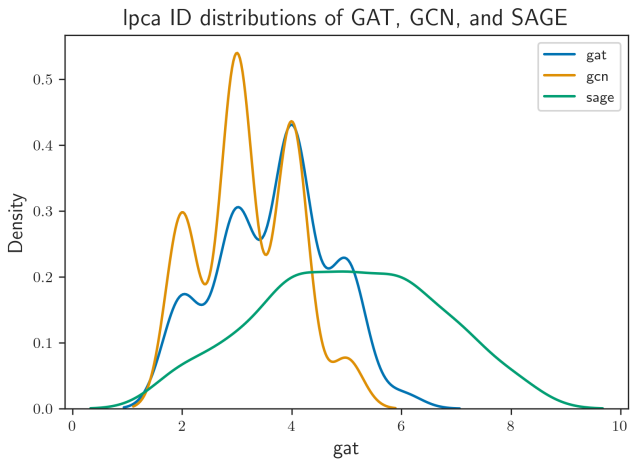

Figure 4.1: Distribution of graph level ID by embedding algorithm: SBM dataset. The visualization shows how embedding IDs vary by embedding algorithm via their kernel density estimates. We have used a common denominator of the *lpca* algorithm as an ID estimator for graph embeddings. We have used the Gaussian kernel as a density estimator.

Table 4.1: Statistically significant correlations for SBM dataset at graph level. The analysis column of the table highlights the nature of a particular analysis task e.g. "Graph IDs & Node Class." indicates that the analysis aims to study the correlation between Graph IDs and the evaluation metrics of Node Classification. We see several significant correlations between graph IDs and performance metrics for node classification and anomaly detection tasks. Notably, natural community size (nclen), NC-LID, and GeoL are associated with model performance (e.g., sage_test_acc, gcn_test_acc). These findings suggest that the intrinsic dimensionality of graphs can impact the effectiveness of machine learning models on graph-structured data.

Analysis	**Variable 1**	**Variable 2**	**Correlation**	**p-value**
Graph IDs & Node Class.	avg_nclid_graph	sage_test_acc	-0.33	0.00
Graph IDs & Anomaly Det.	dim_graph_geol	avg_precision_score	-0.31	0.00
Graph IDs & Node Class.	avg_nclen	sage_test_acc	0.31	0.00
Graph IDs & Anomaly Det.	avg_nclid_graph	avg_precision_score	-0.28	0.00
Graph IDs & Node Class.	avg_nclid_graph	gcn_test_acc	-0.25	0.00
Graph IDs & Node Class.	avg_nclid_graph	gat_test_acc	-0.22	0.00
Graph IDs & Node Class.	avg_nclen	gat_test_acc	0.21	0.00
Graph IDs & Node Class.	dim_graph_geol	gcn_test_acc	-0.21	0.00
Graph IDs & Anomaly Det.	dim_graph_geol	roc_auc	-0.2	0.01
Graph IDs & Anomaly Det.	avg_nclid_graph	roc_auc	-0.19	0.01
Graph IDs & Node Class.	dim_graph_geol	sage_test_acc	-0.18	0.01
Graph IDs & Node Class.	avg_nclen	gcn_test_acc	0.17	0.02

Table 4.2: Node level correlations: SBM dataset. The node level granularity indicates that each node contributes an observation point in the analysis. The table looks at nodes across all the graphs together. It shows a weak negative correlation between node ID (from NCLID) and classification performance for all three models (GAT, GCN, SAGE). Conversely, a positive correlation exists between node's natural community size (nclen) (from NCLID) and classification performance for all models. These results indicate that node-level features, such as ID and nclen, are associated with classification outcomes, though the effects are relatively small compared to the graph-level correlations.

Variable 1	**Variable 2**	**Pearson Corr.**	**P-value**
GAT classification	ID	-0.0335	0.0000
GAT classification	nclen	0.0395	0.0000
GCN classification	ID	-0.0780	0.0000
GCN classification	nclen	0.0914	0.0000
SAGE classification	ID	-0.0540	0.0000
SAGE classification	nclen	0.0659	0.0000

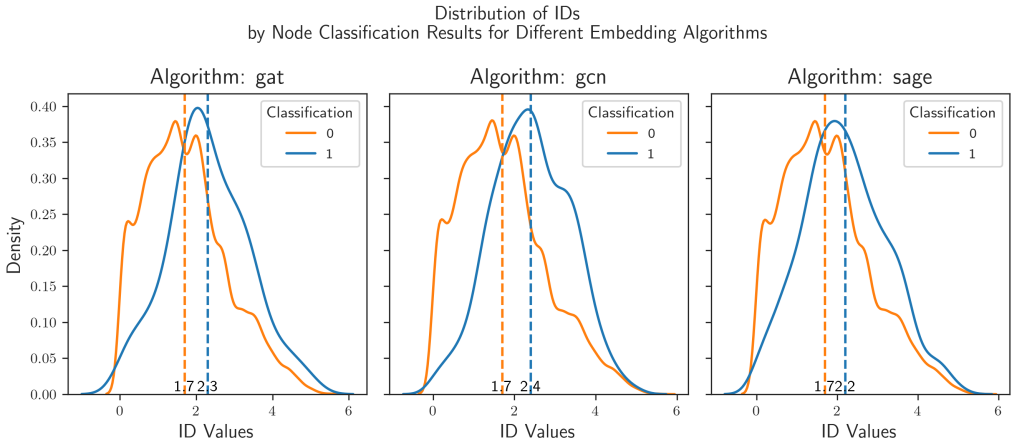

Figure 4.2: Distribution of node level NC-LID by embedding algorithm: SBM graphs. The visualization shows how the accuracy of node classification varies per node, per embedding algorithm, by its NC-LID. The node level granularity indicates that each node contributes an observation point in the analysis. The figure displays kernel density plots of NC-LID values categorized by node classification accuracy (0: wrong predictions, and 1: accurate predictions per node) for three embedding algorithms: GAT, GCN, and SAGE, for the SBM graphs. GAT and GCN show better separation based on NC-LID values with a left-skewed distribution for class 0 and a more spread-out distribution for class 1. All three embedding algorithms exhibit overlapping distributions, suggesting NC-LID values alone may not accurately distinguish accurate node classification. Overall, the figure indicates a potential relationship between NC-LID values and node classification accuracy.

Synthetic graphs: RPSBM

Table 4.3: Statistically significant correlations for RPSBM dataset: graph level. The graph level granularity indicates that each graph contributes an observation point in the analysis. The analysis column of the table highlights the nature of a particular analysis task e.g. "Graph IDs & Node Class." indicates that the analysis aims to study the correlation between Graph IDs and the evaluation metrics of Node Classification. We have found a total of 11 statistically significant correlations. A detailed interpretation is to follow in the Discussion section.

Analysis	**Variable 1**	**Variable 2**	**Correlation**	**p-value**
Embed. IDs & Graph Met.	gat	node_homophily_ratio	-0.14	0.02
Graph IDs & Embed. IDs	dim_graph_geol	gcn	-0.15	0.02
Graph IDs & Node Class.	avg_nclid_graph	gcn_acc	-0.40	0.00
Graph IDs & Node Class.	dim_graph_geol	sage_acc	-0.35	0.00
Graph IDs & Node Class.	graph_nclen	gcn_acc	-0.31	0.00
Graph IDs & Node Class.	graph_nclen	sage_acc	0.31	0.00
Graph IDs & Node Class.	avg_nclid_graph	gat_acc	-0.25	0.00
Graph IDs & Node Class.	avg_nclid_graph	sage_acc	0.19	0.00
Graph IDs & Node Class.	graph_nclen	gat_acc	-0.18	0.00
Graph IDs & Node Class.	dim_graph_geol	gcn_acc	-0.14	0.02

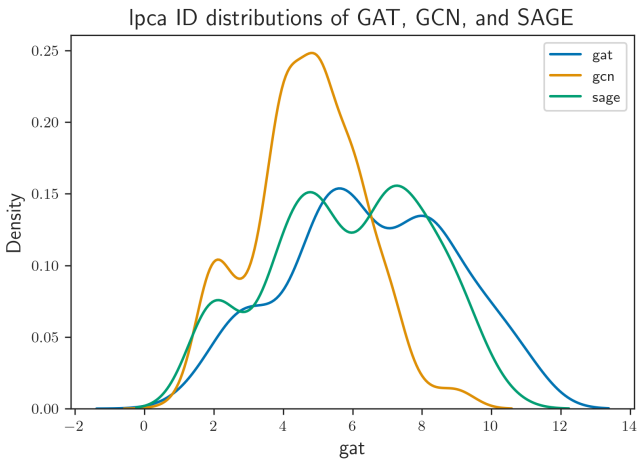

Figure 4.3: Distribution of graph level ID by embedding algorithm: RPSBM dataset. The visualization shows how embedding IDs vary by embedding algorithm via their kernel density estimates. We have used a common denominator of the *lpca* algorithm as an ID estimator for graph embeddings. We have used the Gaussian kernel as a density estimator.

Real graphs

Since there are only 7 graphs in this dataset, we present only the node-level correlations between some key measures.

Table 4.4: Node-level correlations for real graphs' dataset. The node level granularity indicates that each node contributes an observation point in the analysis. The table looks at nodes across all the graphs together. The table examines the relationships between node ID (NC-LID), natural community size (nclen), and node classification performance (accurate or wrong predictions) using three different embedding algorithms (GAT, GCN, SAGE). SAGE node classification shows an interesting negative and statistically significant correlation with node ID.

Variable 1	**Variable 2**	**Pearson Corr.**	**P-value**
GAT classification	ID	-0.0017	0.6426
GAT classification	nclen	-0.0124	0.0010
GCN classification	ID	-0.0398	0.0000
GCN classification	nclen	-0.0056	0.1406
SAGE classification	ID	-0.1163	0.0000
SAGE classification	nclen	0.0580	0.0000

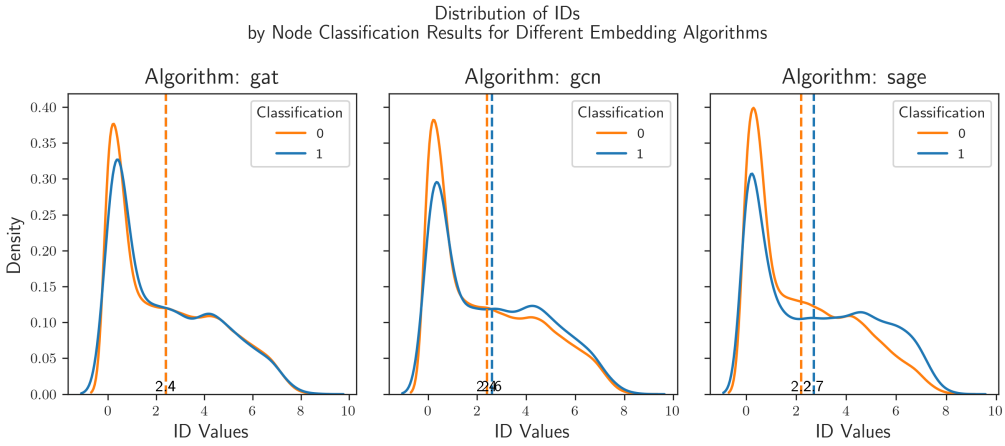

Figure 4.4: Distribution of node level NC-LID by embedding algorithm: real graphs. The visualization shows how the accuracy of node classification varies per node, per embedding algorithm, by its NC-LID. The node level granularity indicates that each node contributes an observation point in the analysis. The figure displays kernel density plots of NC-LID values categorized by node classification accuracy (0: wrong predictions, and 1: accurate predictions per node) for three embedding algorithms: GAT, GCN, and SAGE, for the real graphs. SAGE shows a better separation based on NC-LID values with a left-skewed distribution for both kinds of predictions. All three embedding algorithms exhibit overlapping distributions, suggesting NC-LID values alone may not accurately distinguish accurate node classification. Overall, the figure does not indicate a strong relationship between NC-LID and the accuracy of node classification especially not as strong as displayed in Figure 4.2 corresponding to the study of SBM graphs.

4.4 Discussion

This section addresses the research questions mentioned in section 1.

1. **Is their any correlation between Graph IDs and Graph Embedding IDs?**: We have observed a weak negative linear correlation (see table 4.3, row 2) found between graph ID from GEOL estimator (Stubbemann et al., 2023) and GCN's Node Embedding IDs from mind_ML estimator (Rozza et al., 2012) on the set of RPSBM graphs. No significant correlations were found between other Graph IDs and Embedding IDs.

2. **How do different graph embedding methods affect ID of the resulting representations?**: We have studied the distributions of embedding IDs by embedding algorithm. We have done this by using lpca algorithm to estimate ID of graph embeddings. Figure 4.1 shows different kernel density estimates for the three algorithms, for SBM graphs. Similarly, figure 4.3 shows different kernel density estimates for the three algorithms, for RPSBM graphs. Both the figures showcase different distributions of embedding IDs across embedding algorithms. However, we also observe that while the distributions show promise across both the SBM and RPSBM graph datasets, the nature of the distributions varies significantly. While GCN graph representation has the lowest average resulting ID for both the RPSBM and SBM datasets, SAGE has a notably more spread out distribution in the SBM dataset. These results indicate promising signs that embedding methods influence the resulting representations, however, a detailed study is needed to comment on the nature and extent of this influence.

3. **How do graph metrics impact graph embedding IDs?**: We have observed a weak negative linear correlation found (see table 4.3, row 1) between the graph metric - Node Homophily Ratio (explained in section 3.2) and the GCN's Node Embedding IDs from mind_ML estimator (Rozza et al., 2012) on the set of RPSBM graphs. No significant correlations were found between other graph metrics and Node Embeddings' IDs.

4. **How do graph IDs and embedding IDs impact downstream tasks?**: Results have shown interesting relationships between graph IDs and downstream tasks. Negative weak linear correlations are found between Graph IDs (from both estimators: GEOL and NC-LID) and test accuracy of node classification task performed by all three GNNs:

GraphSAGE, GCN and GAT. Moreover, a negative weak linear correlation is found between Graph IDs (from both estimators: GEOL and NC-LID) and the average precision score of anomaly detection. No significant correlations are found between Embeddings' IDs and the performance of downstream tasks.

5. **How to approach synthetic graph generation for detailed analysis?**: We have observed that to successfully establish a meaningful study between graph IDs, graph metrics, and IDs of graph representations, we need to utilize various graph ID estimators, and graph representation methods on a larger collection of graphs. Regarding graph dataset collection, we need to take care of the following two factors:

   (a) The collection of graphs with a wide range of sizes and parameters such as number of nodes, number of edges, average degree, and node centrality measures.

   (b) Since we need to input these graphs to PyTorch-Geometric libraries to output graph representations, we should be able to convert the generated or collected graphs to *Torch.Data* format[5]. If *Torch.Data* does not receive correct input parameters such as node features, edge list and node labels, we do not get the graph in the correct format. Many of the available torch datasets (example TUDataset [6]) don't have either node features or node labels which means that they cannot be used as input to *torch-geometric.nn* models[7].

Overall, the influence of graph IDs on the quality of downstream machine-learning tasks is emerging as an interesting outcome. The distributions of node ID by whether a node was accurately classified indicate significant differences for all embedding techniques for the SBM graphs.

4.5 Limitations and Future Work

Edge Embeddings We know that nodes and node metrics are crucial part of the graphs to understand innate structures. However, edges have not been studied enough to make any claims. We need to do experiments to understand how graph IDs can impact edge embeddings. Moreover, edge

[5]https://pytorch-geometric.readthedocs.io/en/latest/generated/torch˙geometric.data.Data.html
[6]https://pytorch-geometric.readthedocs.io/en/latest/generated/torch˙geometric.datasets.TUDataset.html
[7]https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html

representations (S. Liu et al., 2024) need to be experimented with to understand their impact on downstream tasks such as outlier edges, community detection and graph reconstruction.

More graph ID estimators:
Some ID estimators that are available for non-euclidean spaces and complex networks, can be applied to graphs (Macocco et al., 2024). It needs an in-depth study to formulate and modify these ID estimators for graph datasets. Moreover, implementations of some algorithms are also not openly available (Granata and Carnevale, 2016).

Upcoming graph representation methods:
There have been recent upcoming novel algorithms for graph representations, which can be utilized to conduct an exhaustive study. Some interesting Graph Embedding methods to look out for: WalkLM (Tan et al., 2023), SDFormer (D. Li et al., 2024), (S. Liu et al., 2024), and Language Model-Based Knowledge Graph Embeddings (Cheng et al., 2024).

More real graphs:
As the study has only inspected seven real graphs in comparison to 450 synthetic graphs, it leaves significant room to explore the research questions on a larger volume of real graph structures.

In conclusion, this research lays the groundwork for a deeper understanding of graph representations and their impacts on machine learning tasks, highlighting both the potential and the need for continued investigation in this evolving field.

Chapter 5

Conclusion

This study explored the relationship between graph IDs and graph embeddings and their impact on various machine-learning tasks. The findings contribute valuable insights into how different graph representation methods influence the resulting representations and their utilities in downstream applications. The study sets a baseline for the addressed research questions and provides a framework to (a) conduct such experiments, especially with synthetic graph datasets, and (b) measure impact. The p-values (6.1) show that these small/weak correlations are with high confidence.

Additionally, the study highlights the nuances of graph structures. This thesis provides a baseline measure and a scientific methodology to inspect these structures and shows how much the natural/innate complexities (ID) translate into graph repsentations. Future work should focus on:

- Expanding the range of graph metrics and embedding methods explored.
- Investigating the impact of edge embeddings and additional graph-wide measures.
- Applying and adapting advanced graph ID estimators for complex network datasets.
- Exploring other downstream tasks, such as graph classification, edge outlier detection, to evaluate broader applicability.

Bibliography

Amsaleg, L., Chelly, O., Furon, T., Girard, S., Houle, M. E., Kawarabayashi, K.-i., & Nett, M. (2015). Estimating local intrinsic dimensionality. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 29–38.

Anderson, C. J., Wasserman, S., & Faust, K. (1992). Building stochastic blockmodels. *Social networks*, *14*(1-2), 137–161.

Ansuini, A., Laio, A., Macke, J. H., & Zoccolan, D. (2019). Intrinsic dimension of data representations in deep neural networks. *CoRR*, *abs/1905.12784*. http://arxiv.org/abs/1905.12784

Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *science*, *286*(5439), 509–512.

Bojchevski, A., & Günnemann, S. (2018). Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. https://arxiv.org/abs/1707.03815

Bollobás, B. (1998). Random graphs. In *Modern graph theory* (pp. 215–252). Springer New York. https://doi.org/10.1007/978-1-4612-0619-4_7

Bowers, A. J., & Zhou, X. (2019). Receiver operating characteristic (roc) area under the curve (auc): A diagnostic measure for evaluating the accuracy of predictors of education outcomes. *Journal of Education for Students Placed at Risk (JESPAR)*, *24*(1), 20–46.

Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, *25*(2), 163–177.

Brody, S., Alon, U., & Yahav, E. (2021). How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*.

Ceruti, C., Bassis, S., Rozza, A., Lombardi, G., Casiraghi, E., & Campadelli, P. (2012). Danco: Dimensionality from angle and norm concentration. *CoRR*, *abs/1206.3881*. http://arxiv.org/abs/1206.3881

Chen, F., Wang, Y.-C., Wang, B., & Kuo, C.-C. J. (2020). Graph representation learning: A survey. *APSIPA Transactions on Signal and Information Processing*, *9*, e15. https://doi.org/10.1017/ATSIP.2020.13

Chen, T., Bian, S., & Sun, Y. (2019). Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579*.

Chen, Z., Li, X., & Bruna, J. (2017). Supervised community detection with line graph neural networks. *arXiv preprint arXiv:1705.08415*.

Cheng, S., Zhang, N., Tian, B., Chen, X., Liu, Q., & Chen, H. (2024). Editing language model-based knowledge graph embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(16), 17835–17843.

Collobert, R., Bengio, S., & Mariéthoz, J. (2002). Torch: A modular machine learning software library.

Costa, J. A., Girotra, A., & Hero, A. (2005). Estimating local intrinsic dimension with k-nearest neighbor graphs. *IEEE/SP 13th Workshop on Statistical Signal Processing, 2005*, 417–422.

Csardi, M. G. (2013). Package 'igraph'. *Last accessed*, *3*(09), 2013.

Ding, K., Li, J., Bhanushali, R., & Liu, H. (n.d.). Deep anomaly detection on attributed networks. In *Proceedings of the 2019 siam international conference on data mining (sdm)* (pp. 594–602). https://doi.org/10.1137/1.9781611975673.67

Fan, M., Gu, N., Qiao, H., & Zhang, B. (2010). Intrinsic dimension estimation of data by principal component analysis. *CoRR*, *abs/1002.2050*. http://arxiv.org/abs/1002.2050

Farahmand, A. m., Szepesvári, C., & Audibert, J.-Y. (2007). Manifold-adaptive dimension estimation. *Proceedings of the 24th International Conference on Machine Learning*, 265–272. https://doi.org/10.1145/1273496.1273530

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. https://arxiv.org/abs/1903.02428

Fisher, R. (1925). *Statistical methods for research workers, 11th ed. rev.* Edinburgh.

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, *222*(594-604), 309–368.

Freeman, L. C., et al. (2002). Centrality in social networks: Conceptual clarification. *Social network: critical concepts in sociology. Londres: Routledge*, *1*, 238–263.

Fukunaga, K., & Olsen, D. (1971). An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, *C-20*(2), 176–183. https://doi.org/10.1109/T-C.1971.223208

Goh, K.-I., Kahng, B., & Kim, D. (2001). Universal behavior of load distribution in scale-free networks. *Physical review letters*, *87*(27), 278701.

Granata, D., & Carnevale, V. (2016). Accurate estimation of the intrinsic dimension using graph distances: Unraveling the geometric complexity of datasets. *Scientific reports*, *6*(1), 31377.

Grassberger, P., & Procaccia, I. (1983). Measuring the strangeness of strange attractors. *Physica D: Nonlinear Phenomena*, *9*(1-2), 189–208.

Greenland, S., Senn, S. J., Rothman, K. J., Carlin, J. B., Poole, C., Goodman, S. N., & Altman, D. G. (2016). Statistical tests, P values, confidence intervals, and power: A guide to misinterpretations. *Eur J Epidemiol*, *31*(4), 337–350.

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. *CoRR*, *abs/1607.00653*. http://arxiv.org/abs/1607.00653

Guyon, I., Gunn, S., Ben-Hur, A., & Dror, G. (2004). Result analysis of the nips 2003 feature selection challenge. *Advances in neural information processing systems*, *17*.

Hagberg, A., Swart, P. J., & Schult, D. A. (2008). *Exploring network structure, dynamics, and function using networkx* (tech. rep.). Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).

Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, *30*.

Heist, N., Hertling, S., & Paulheim, H. (2023). Kgreat: A framework to evaluate knowledge graphs via downstream tasks. *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 3938–3942. https://doi.org/10.1145/3583780.3615241

Holme, P., & Kim, B. J. (2002). Growing scale-free networks with tunable clustering. *Phys. Rev. E*, *65*, 026107. https://doi.org/10.1103/PhysRevE.65.026107

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, *24*(6), 417.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Kim, D., & Oh, A. (2021). How to find your friendly neighborhood: Graph attention design with self-supervision. *International Conference on Learning Representations*. https://openreview.net/forum?id=Wi5KUNlqWty

Kipf, T. N., & Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kipf, T. N., & Welling, M. (2016b). Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

Lamsal, S. (2024). Graph neural networks for outlier detection.

Lancichinetti, A., Fortunato, S., & Kertész, J. (2009). Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, *11*(3), 033015. https://doi.org/10.1088/1367-2630/11/3/033015

Levina, E., & Bickel, P. (2004). Maximum likelihood estimation of intrinsic dimension. *Advances in neural information processing systems*, *17*.

Li, C., Farkhoor, H., Liu, R., & Yosinski, J. (2018). Measuring the intrinsic dimension of objective landscapes. *CoRR*, *abs/1804.08838*. http://arxiv.org/abs/1804.08838

Li, D., Xia, T., Wang, J., Shi, F., Zhang, Q., Li, B., & Xiong, Y. (2024). Sdformer: A shallow-to-deep feature interaction for knowledge graph embedding. *Knowledge-Based Systems*, *284*, 111253. https://doi.org/https://doi.org/10.1016/j.knosys.2023.111253

Liu, K., Dou, Y., Zhao, Y., Ding, X., Hu, X., Zhang, R., Ding, K., Chen, C., Peng, H., Shu, K., Sun, L., Li, J., Chen, G. H., Jia, Z., & Yu, P. S. (2022). Bond: Benchmarking unsupervised outlier node detection on static attributed graphs. https://arxiv.org/abs/2206.10071

Liu, S., Yao, D., Fang, L., Li, Z., Li, W., Feng, K., Ji, X., & Bi, J. (2024). Anomalyllm: Few-shot anomaly edge detection for dynamic graphs using large language models. *arXiv preprint arXiv:2405.07626*.

Macocco, I., Mira, A., & Laio, A. (2024). Intrinsic dimension as a multi-scale summary statistics in network modeling. *Scientific Reports*, *14*(1), 17756.

McKinney, W., et al. (2011). Pandas: A foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, *14*(9), 1–9.

Mo, D., & Huang, S. H. (2012). Fractal-based intrinsic dimension estimation and its application in dimensionality reduction. *IEEE Transactions on Knowledge and Data Engineering*, *24*(1), 59–71. https://doi.org/10.1109/TKDE.2010.225

Newman, M., & Watts, D. (1999). Renormalization group analysis of the small-world network model. *Physics Letters A*, *263*(4), 341–346. https://doi.org/https://doi.org/10.1016/S0375-9601(99)00757-4

Neyman, J., Pearson, E. S., & Pearson, K. (1933). Ix. on the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, *231*(694-706), 289–337. https://doi.org/10.1098/rsta.1933.0009

Obi, J. C. (2023). A comparative study of several classification metrics and their performances on data. *World Journal of Advanced Engineering Technology and Sciences*, *8*(1), 308–314.

Oliphant, T. E., et al. (2006). *Guide to numpy* (Vol. 1). Trelgol Publishing USA.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

Pearson, K. (1894). Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, *185*, 71–110.

Pearson, K. (1895). Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, *58*, 240–242. Retrieved August 11, 2024, from http://www.jstor.org/stable/115794

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. *CoRR*, *abs/1403.6652*. http://arxiv.org/abs/1403.6652

Pestov, V. (2007). Intrinsic dimension of a dataset: What properties does one expect? *2007 International Joint Conference on Neural Networks*, 2959–2964.

Procházka, P., Mares, M., & Dedic, M. (2023). Which graph properties affect gnn performance for a given downstream task? *ITAT*, 58–66.

Rozza, A., Lombardi, G., Ceruti, C., Casiraghi, E., & Campadelli, P. (2012). Novel high intrinsic dimensionality estimators. *Machine learning*, *89*, 37–65.

Savić, M., Kurbalija, V., & Radovanović, M. (2023). Local intrinsic dimensionality measures for graphs, with applications to graph embeddings. *Information Systems*, *119*, 102272. https://doi.org/https://doi.org/10.1016/j.is.2023.102272

Somorjai, R. (1986). Methods for estimating the intrinsic dimsnionality of high-dimensional point sets. In *Dimensions and entropies in chaotic systems: Quantification of complex behavior* (pp. 137–147). Springer.

Stubbemann, M., Hanika, T., & Schneider, F. M. (2023). Intrinsic dimension for large-scale geometric learning. https://arxiv.org/abs/2210.05301

Student. (1908). The probable error of a mean. *Biometrika*, *6*(1), 1–25. Retrieved August 11, 2024, from http://www.jstor.org/stable/2331554

Tan, Y., Zhou, Z., Lv, H., Liu, W., & Yang, C. (2023). Walklm: A uniform language model fine-tuning framework for attributed graph embed-

ding. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 13308–13325, Vol. 36). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2023/file/2ac879d1865475a7abc8dfc7a9c15c27-Paper-Conference.pdf

Tenenbaum, J. B., Silva, V. d., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *science*, *290*(5500), 2319–2323.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, N., Lin, L., Li, J., & Wang, H. (2022). Unbiased graph embedding with biased graph observations. *Proceedings of the ACM Web Conference 2022*, 1423–1433. https://doi.org/10.1145/3485447.3512189

Wang, X., Slavakis, K., & Lerman, G. (2015). Multi-manifold modeling in non-euclidean spaces. *Artificial Intelligence and Statistics*, 1023–1032.

Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, *2*(1-3), 37–52.

Xiao, S., Wang, S., Dai, Y., & Guo, W. (2022). Graph neural networks in node classification: Survey and evaluation. *Machine Vision and Applications*, *33*(1), 4.

Xu, M. (2021). Understanding graph embedding methods and their applications. *SIAM Review*, *63*(4), 825–853.

Yang, Z., Cohen, W. W., & Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. https://arxiv.org/abs/1603.08861

Zhang, M., Li, P., Xia, Y., Wang, K., & Jin, L. (2020). Revisiting graph neural networks for link prediction.

Zhou, Y., Zheng, H., Huang, X., Hao, S., Li, D., & Zhao, J. (2022). Graph neural networks: Taxonomy, advances, and trends. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *13*(1), 1–54.

Chapter 6

Appendix

6.1 Definition of Pearson's Correlation Coefficient

Pearson's product-moment correlation coefficient (commonly denoted as r) quantifies the linear association between two continuous variables, X and Y, assumed to be measured on an interval or ratio scale (Pearson, 1895). It is calculated as:

$$r = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \tag{6.1}$$

where:

- $\text{cov}(X, Y)$ is the covariance of X and Y
- σ_X and σ_Y are the standard deviations of X and Y
- $\mathbb{E}$ is the expectation operator
- μ_X and μ_Y are the means of X and Y

The coefficient ranges from -1 to +1:

- $r = +1$: Perfect positive linear relationship.
- $r = -1$: Perfect negative linear relationship.
- $r = 0$: No linear relationship (though other relationships may exist).

For a sample of n paired observations (x_i, y_i), the sample correlation coefficient is:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{6.2}$$

where $\bar{x}$ and $\bar{y}$ are the sample means.

It's crucial to note that Pearson's correlation measures only linear relationships and can be influenced by outliers.

Significance Testing of Pearson's Correlation Coefficient

When we calculate a Pearson correlation coefficient, r, we want to know if the observed correlation happened by chance or represents a real relationship in the population. To determine this, we use a significance test, a statistical procedure rooted in the work of Neyman and Pearson (Neyman et al., 1933). This test helps us decide whether the observed correlation is statistically significant.

1. **Hypotheses**: We start with two opposing statements:
   - **Null Hypothesis ($\mathbf{H_0}$)**: There is no linear correlation in the population ($\rho = 0$).
   - **Alternative Hypothesis ($\mathbf{H_1}$)**: There is a linear correlation in the population ($\rho \neq 0$). We use a two-sided test to consider both positive and negative correlations.

2. **Test Statistic**: For large sample sizes, we use a t-statistic to test our hypothesis. Under the null hypothesis, this statistic follows a t-distribution with $n-2$ degrees of freedom (Student, 1908). This distribution, developed by William Sealy Gosset (published as "Student"), is commonly used in hypothesis testing. We calculate the t-statistic as follows:

   $$t = r\sqrt{\frac{n-2}{1-r^2}} \tag{6.3}$$

   where r is the sample correlation coefficient and n is the number of observations.

3. **P-value**: The p-value, a concept developed by Fisher (R. Fisher, 1925), tells us the probability of getting a test statistic as extreme as the one we observed or even more extreme if the null hypothesis were true (Greenland et al., 2016). A lower p-value means the observed correlation is less likely to occur by chance alone.

4. **Decision Rule**: We set a significance level, usually $\alpha = 0.05$, as a threshold. If the p-value is less than α, we reject the null hypothesis, concluding that the correlation is statistically significant. If the p-value is greater than or equal to α, we do not have enough evidence to reject the null hypothesis.

5. **Interpretation**: The p-value alone doesn't tell us everything. We also need to consider the strength of the correlation (the magnitude of r). A small p-value only tells us that the correlation is unlikely to be zero.

Importantly, this test relies on certain assumptions, such as the data following a bivariate normal distribution. If these assumptions are violated, the results might not be accurate.

6.2 Evaluation metrics for downstream tasks: ROC-AUC, precision score and accuracy

ROC Stands for "Receiver Operating Characteristic" and AUC stands for "Area under the ROC Curve" (Bowers and Zhou, 2019). To create a ROC curve, we need to plot "True Positive Rate" vs "False Positive Rate".

TP is the number of True Positives, FN is the number of False Negatives, FP is the number of False positives and TN is the number of True Negatives.

Recall (True Positive Rate) is defined as the ratio of actual (true) positives identified by the classifier,
$TPR = TP/(TP + FN)$
Precision is defined as the correct number of actual positives out of all the positives identified by the classifier,
$precision = TP/(TP + FP)$

False Positive Rate (FPR) signifies actual negatives incorrectly identified by the classifier:
$FPR = FP/(TN + FP)$

Accuracy is defined as:

$Accuracy = (TP + TN)/(TP + FP + FN + TN)$

How to interpret AUC:

AUC = 1.0, means a perfect classifier,
AUC = 0.5, no significant value provided by the classifier.
$AUC < 0.5$, worse performance, random guessing is considered even better.
The AUC score of the link prediction task measures how well the model identifies the correct links (links/edges that should exist) and incorrect links (links that should not exist).

The anomaly detection model is a binary model that labels normal nodes as (0) and anomalous or outlier nodes as (1). The AUC score of the Anomaly detection task measures how correctly the model has distinguished the normal and anomalous nodes.

Accuracy defines the correct number of labels the model identifies, an important measure to evaluate node classifier (Obi, 2023). Higher accuracy signifies better performance of the classifier.

6.3 SBM correlations

Table 6.1: Correlations: Graph IDs and Embedding IDs (mind˙ml)

Variable 1	Variable 2	Correlation	p-value
avg_nclen	gat	-0.11	0.14
avg_nclen	sage	-0.11	0.11
avg_nclen	gcn	-0.1	0.15
avg_nclid_graph	gcn	0.07	0.31
dim_graph_geol	gat	-0.06	0.42
avg_nclid_graph	sage	0.03	0.71
avg_nclid_graph	gat	0.02	0.77
dim_graph_geol	gcn	0.02	0.83
dim_graph_geol	sage	0	0.98

Table 6.2: Correlations: Embedding IDs (mind ml) and Graph metrics

Variable 1	Variable 2	Correlation	p-value
gcn	num_classes	0.12	0.1
sage	close_cent	-0.11	0.12
gcn	node_features	0.09	0.23
gat	node_features	-0.09	0.19
gat	between_cent	0.08	0.28
gat	load_cent	0.08	0.28
gat	degree_cent	0.06	0.41
gcn	close_cent	-0.06	0.44
gat	close_cent	-0.05	0.47
sage	load_cent	0.05	0.49
gcn	num_nodes	0.05	0.45
sage	between_cent	0.05	0.49
gcn	num_edges	0.04	0.59
gat	num_nodes	-0.03	0.63
sage	num_nodes	-0.02	0.8
gat	num_classes	-0.02	0.75
gcn	between_cent	-0.02	0.78
gcn	load_cent	-0.02	0.78
gcn	degree_cent	0.02	0.79
sage	degree_cent	0.02	0.81
sage	num_classes	-0.02	0.78
sage	node_features	-0.01	0.85
sage	num_edges	-0.01	0.87
gat	num_edges	-0.01	0.85

Table 6.3: Correlations: Graph IDs and Graph Metrics

Variable 1	Variable 2	Correlation	p-value
avg_nclen	num_classes	0.12	0.09
avg_nclen	num_edges	0.08	0.27
dim_graph_geol	degree_cent	-0.07	0.36
avg_nclen	num_nodes	0.07	0.32
avg_nclen	degree_cent	0.06	0.43
avg_nclid_graph	load_cent	-0.05	0.46
avg_nclid_graph	between_cent	-0.05	0.46
avg_nclid_graph	close_cent	0.05	0.49
avg_nclen	close_cent	0.04	0.59
avg_nclid_graph	node_features	0.04	0.54
dim_graph_geol	between_cent	-0.04	0.62
dim_graph_geol	load_cent	-0.04	0.62
avg_nclid_graph	degree_cent	-0.03	0.67
avg_nclid_graph	num_edges	0.03	0.63
dim_graph_geol	node_features	0.03	0.64
avg_nclen	between_cent	-0.03	0.64
avg_nclen	load_cent	-0.03	0.64
dim_graph_geol	num_classes	-0.03	0.65
dim_graph_geol	close_cent	-0.03	0.65
avg_nclid_graph	num_nodes	0.02	0.8
avg_nclid_graph	num_classes	-0.02	0.74
dim_graph_geol	num_edges	-0.02	0.83
dim_graph_geol	num_nodes	0.01	0.87
avg_nclen	node_features	0.01	0.84

Table 6.4: Correlations: Graph IDs and Node Class.

Variable 1	Variable 2	Correlation	p-value
avg_nclid_graph	sage_test_acc	-0.33	0
avg_nclen	sage_test_acc	0.31	0
avg_nclid_graph	gcn_test_acc	-0.25	0
avg_nclid_graph	gat_test_acc	-0.22	0
avg_nclen	gat_test_acc	0.21	0
dim_graph_geol	gcn_test_acc	-0.21	0
dim_graph_geol	sage_test_acc	-0.18	0.01
avg_nclen	gcn_test_acc	0.17	0.02
dim_graph_geol	gat_test_acc	0	0.95

Table 6.5: Correlations: Graph IDs and Link Pred.

Variable 1	Variable 2	Correlation	p-value
avg_nclen	test_auc	0.08	0.3
avg_nclid_graph	test_auc	0.06	0.43
dim_graph_geol	test_auc	-0.05	0.52

Table 6.6: Correlations: Graph IDs and Anomaly Detection Metrics

Variable 1	Variable 2	Correlation	p-value
dim_graph_geol	avg_precision_score	-0.31	0
avg_nclid_graph	avg_precision_score	-0.28	0
dim_graph_geol	roc_auc	-0.2	0.01
avg_nclid_graph	roc_auc	-0.19	0.01
avg_nclen	roc_auc	0.01	0.85
avg_nclen	avg_precision_score	0	0.97

Table 6.7: Correlations: Graph Metrics and Node Class.

Variable 1	Variable 2	Correlation	p-value
node_features	gat_test_acc	0.13	0.06
degree_cent	gcn_test_acc	-0.12	0.11
close_cent	gat_test_acc	-0.12	0.1
degree_cent	gat_test_acc	-0.12	0.11
num_classes	sage_test_acc	0.12	0.11
close_cent	sage_test_acc	-0.11	0.14
num_classes	gcn_test_acc	0.11	0.13
num_nodes	gcn_test_acc	0.1	0.16
num_nodes	sage_test_acc	0.09	0.24
close_cent	gcn_test_acc	-0.09	0.21
node_features	sage_test_acc	0.09	0.2
node_features	gcn_test_acc	0.08	0.28
degree_cent	sage_test_acc	-0.08	0.29
load_cent	gcn_test_acc	-0.07	0.33
num_classes	gat_test_acc	0.07	0.34
load_cent	sage_test_acc	-0.07	0.33
between_cent	gcn_test_acc	-0.07	0.33
num_nodes	gat_test_acc	0.07	0.32
between_cent	sage_test_acc	-0.07	0.33
num_edges	gcn_test_acc	0.04	0.63
num_edges	sage_test_acc	0.03	0.63
load_cent	gat_test_acc	-0.02	0.77
between_cent	gat_test_acc	-0.02	0.77
num_edges	gat_test_acc	0.02	0.74

Table 6.8: Correlations: Graph Metrics and Link Pred.

Variable 1	Variable 2	Correlation	p-value
num_edges	test_auc	0.05	0.51
between_cent	test_auc	-0.05	0.45
load_cent	test_auc	-0.05	0.45
num_classes	test_auc	0.04	0.57
num_nodes	test_auc	0.03	0.7
node_features	test_auc	-0.02	0.75
degree_cent	test_auc	0.02	0.82
close_cent	test_auc	0	0.97

Table 6.9: Correlations: Graph Metrics and Anomaly Detection Metrics

Variable 1	Variable 2	Correlation	p-value
avg_nclen	test_auc	0.08	0.3
avg_nclid_graph	test_auc	0.06	0.43
dim_graph_geol	test_auc	-0.05	0.52

6.4 RPSBM correlations

Table 6.10: Graph IDs and Embedding IDs

Variable 1	Variable 2	Correlation	p-value
graph_nclen	gcn	0.10	0.12
graph_nclen	gat	0.07	0.24
graph_nclen	sage	0.07	0.25
dim_graph_geol	sage	0.05	0.46
dim_graph_geol	gat	0.05	0.46
avg_nclid_graph	sage	0.03	0.68
avg_nclid_graph	gat	0.02	0.73
avg_nclid_graph	gcn	0.01	0.89

Table 6.11: Graph IDs and Graph Metrics

Variable 1	Variable 2	Correlation	p-value
avg_nclid_graph	avg_degree	0.12	0.05
avg_nclid_graph	num_edges	0.12	0.05
graph_nclen	num_edges	0.11	0.07
graph_nclen	avg_degree	0.11	0.09
dim_graph_geol	degree_cent	-0.10	0.11
avg_nclid_graph	close_cent	0.09	0.17
dim_graph_geol	load_cent	-0.08	0.20
dim_graph_geol	between_cent	-0.08	0.20
avg_nclid_graph	node_features	-0.07	0.25
graph_nclen	close_cent	0.07	0.26
dim_graph_geol	num_classes	0.07	0.27
avg_nclid_graph	node_homophily_ratio	-0.06	0.32
graph_nclen	node_features	-0.06	0.36
dim_graph_geol	num_edges	0.05	0.42
dim_graph_geol	avg_degree	0.05	0.46
dim_graph_geol	node_homophily_ratio	0.04	0.55
graph_nclen	num_nodes_per_class	0.03	0.60
avg_nclid_graph	num_classes	-0.03	0.61
graph_nclen	node_homophily_ratio	-0.03	0.62
dim_graph_geol	node_features	0.03	0.63
dim_graph_geol	num_nodes	0.03	0.65
dim_graph_geol	num_nodes_per_class	0.03	0.66
graph_nclen	num_nodes	0.03	0.67
graph_nclen	between_cent	-0.02	0.70
graph_nclen	load_cent	-0.02	0.70
avg_nclid_graph	num_nodes_per_class	0.02	0.75
avg_nclid_graph	degree_cent	0.02	0.80
graph_nclen	degree_cent	0.02	0.80
avg_nclid_graph	load_cent	-0.01	0.85
avg_nclid_graph	between_cent	-0.01	0.85
avg_nclid_graph	num_nodes	0.01	0.88
graph_nclen	num_classes	0.00	0.95
dim_graph_geol	close_cent	0.00	0.97

Table 6.12: Embedding IDs and Graph Metrics

Variable 1	Variable 2	Correlation	p-value
gcn	num_edges	0.12	0.05
gat	degree_cent	-0.12	0.06
gcn	num_nodes_per_class	0.12	0.06
gcn	node_homophily_ratio	-0.11	0.07
sage	node_homophily_ratio	-0.11	0.07
sage	close_cent	0.11	0.09
sage	avg_degree	0.10	0.10
gcn	num_nodes	0.10	0.12
gat	num_edges	0.09	0.17
gat	load_cent	-0.09	0.17
gat	between_cent	-0.09	0.17
gcn	node_features	0.09	0.18
gat	num_classes	0.08	0.21
gat	avg_degree	0.08	0.22
gcn	num_classes	0.08	0.23
gat	num_nodes_per_class	0.06	0.33
sage	node_features	-0.05	0.41
sage	num_nodes_per_class	-0.05	0.43
gat	num_nodes	0.05	0.47
gcn	avg_degree	0.04	0.51
sage	num_nodes	-0.04	0.51
gcn	degree_cent	-0.03	0.59
gat	close_cent	0.03	0.66
sage	num_edges	0.02	0.72
gcn	load_cent	-0.02	0.73
gcn	between_cent	-0.02	0.73
gcn	close_cent	0.01	0.88
sage	load_cent	-0.01	0.88
sage	between_cent	-0.01	0.88
gat	node_features	-0.01	0.92
sage	degree_cent	-0.01	0.92
sage	num_classes	0.00	0.96

Table 6.13: Graph IDs and Anomaly Detection Metrics

Variable 1	Variable 2	Correlation	p-value
avg_nclid_graph	roc_auc	0.12	0.11
dim_graph_geol	roc_auc	-0.10	0.19
avg_nclid_graph	avg_precision_score	0.08	0.30
graph_nclen	roc_auc	0.06	0.44
graph_nclen	avg_precision_score	0.02	0.85
dim_graph_geol	avg_precision_score	0.00	0.95

Table 6.14: Graph IDs and Link Prediction Metrics

Variable 1	Variable 2	Correlation	p-value
dim_graph_geol	test_auc	-0.07	0.28
graph_nclen	test_auc	0.04	0.52
avg_nclid_graph	test_auc	0.00	0.96

Table 6.15: Graph IDs and Node Classification Metrics

Variable 1	Variable 2	Correlation	p-value
dim_graph_geol	gat_acc	-0.12	0.07

Table 6.16: Graph Metrics and Anomaly Detection Metrics

Variable 1	Variable 2	Correlation	p-value
num_nodes_per_class	roc_auc	0.12	0.12
node_features	roc_auc	0.11	0.14
num_nodes	roc_auc	0.08	0.28
between_cent	avg_precision_score	0.08	0.29
load_cent	avg_precision_score	0.08	0.29
num_edges	avg_precision_score	0.08	0.32
node_features	avg_precision_score	0.08	0.33
degree_cent	avg_precision_score	0.07	0.35
num_edges	roc_auc	0.04	0.64
between_cent	roc_auc	-0.04	0.64
load_cent	roc_auc	-0.04	0.64
num_classes	roc_auc	0.04	0.65
avg_degree	avg_precision_score	0.03	0.71
num_nodes	avg_precision_score	0.02	0.78
close_cent	roc_auc	-0.02	0.79
node_homophily_ratio	roc_auc	-0.02	0.81
degree_cent	roc_auc	-0.02	0.82
avg_degree	roc_auc	-0.02	0.83
num_nodes_per_class	avg_precision_score	0.02	0.84
close_cent	avg_precision_score	0.01	0.93
node_homophily_ratio	avg_precision_score	0.01	0.94
num_classes	avg_precision_score	0.00	0.95

Table 6.17: Graph Metrics and Link Prediction Metrics

Variable 1	Variable 2	Correlation	p-value
num_classes	test_auc	-0.13	0.05
num_nodes_per_class	test_auc	-0.12	0.06
num_nodes	test_auc	-0.11	0.09
node_homophily_ratio	test_auc	-0.11	0.09
degree_cent	test_auc	0.06	0.36
close_cent	test_auc	0.04	0.49
between_cent	test_auc	0.03	0.64
load_cent	test_auc	0.03	0.64
avg_degree	test_auc	0.02	0.75
num_edges	test_auc	0.01	0.86

Table 6.18: Graph Metrics and Node Classification Metrics

Variable 1	Variable 2	Correlation	p-value
node_features	gat_acc	0.09	0.15
between_cent	gcn_acc	0.08	0.23
load_cent	gcn_acc	0.08	0.23
node_homophily_ratio	sage_acc	-0.07	0.25
degree_cent	gcn_acc	0.07	0.29
num_edges	gat_acc	-0.06	0.32
num_nodes_per_class	gat_acc	-0.06	0.38
node_features	gcn_acc	0.05	0.39
avg_degree	sage_acc	0.05	0.41
num_edges	gcn_acc	-0.05	0.42
close_cent	sage_acc	0.05	0.45
degree_cent	sage_acc	0.05	0.47
num_nodes	gat_acc	-0.04	0.49
num_classes	gat_acc	-0.04	0.50
load_cent	gat_acc	0.04	0.53
between_cent	gat_acc	0.04	0.53
between_cent	sage_acc	0.04	0.54
load_cent	sage_acc	0.04	0.54
node_features	sage_acc	0.04	0.55
num_classes	sage_acc	0.04	0.57
node_homophily_ratio	gcn_acc	0.03	0.62
num_nodes_per_class	gcn_acc	-0.03	0.62
avg_degree	gat_acc	-0.03	0.66
num_edges	sage_acc	0.03	0.67
num_nodes	gcn_acc	-0.03	0.68
num_nodes	sage_acc	0.02	0.73
node_homophily_ratio	gat_acc	-0.02	0.74
close_cent	gcn_acc	0.02	0.75
degree_cent	gat_acc	0.02	0.75
close_cent	gat_acc	-0.02	0.78
num_nodes_per_class	sage_acc	0.01	0.85
avg_degree	gcn_acc	0.01	0.90
num_classes	gcn_acc	-0.01	0.93