Mandatory Summary

Data is all around us. Every year, millions of petabytes of data are recorded. Machine learning has become an increasingly popular approach to learning from this data, where many of these models and methods assume a tabular data structure. However, a graph-based representation has advantages over a tabular representation, as it can more naturally model data sets with complex relationships and dependencies. Graphs are used in a variety of domains, for example, IoT networks of sensors, traffic networks, and knowledge graphs. The use of Graph Neural Networks (GNN) models has increasingly become popular for machine learning tasks, such as node classification and relation prediction. Models such as the Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT) have seen ground-breaking results in the field of deep learning. Unlike traditional neural networks, these models assume a graph-based representation of the data and can leverage the inherent structural information available in them. GNN models are powerful, but missing values pose a challenge for these models.

Missing values may be viewed as a type of erroneous data, and thus warrants detection and correction measures. Data may be missing for various reasons that may occur during recording or when merging data sets. Samples may be missing due to a sensor malfunction during recording. Participants filling in a questionnaire may decide to leave answers blank. Physicians recording medical tests for patients may decide to leave the results unrecorded if they are not abnormal. As such, missing values are an unavoidable fact of life when dealing with data sets. When modeling the data as graphs, the missingness will manifest as missing nodes, edges, or attributes. Dealing with the missingness problem is important, as it can otherwise greatly impact a model's performance by introducing bias and making it less generalizable. Depending on the underlying reason, missing data can be categorized into three different missingness mechanisms: Missing Completely at Random (MCAR), Missing at Random (MAR), and Not Missing at Random (NMAR).

Previous work has suggested methods to test for the mechanisms of missingness; for example, Gad-Elrab et al. present a method to test for MCAR using Explainable Boosting Machines. However, this method assumes a tabular dataset. In this work, we examine how the mechanism of missingness may be determined using clustering and classification using a sub-graph of Wikidata. The impact of the different missingness mechanisms for downstream machine learning tasks is not a well-studied subject for graphs. While previous research has examined how to handle different missingness mechanisms for tabular data, the same has not been done for graph data. In this work, we simulate missingness based on missingness mechanisms and formulate link prediction as an imputation method. We examine how the downstream task of sepsis prediction, in the MIMIC-IV dataset, is impacted by missingness, and how performance may change with imputation.

Our contribution can, therefore, be summarized as follows.

- We propose a method for simulating missingness mechanisms in a graph, given a specified rate.
- We empirically evaluate how the different mechanisms and missing rates affect downstream tasks for graph machine learning, specifically sepsis prediction.
- We formulate link prediction as an imputation method for missing edges in a graph and investigate if it can improve the downstream task of sepsis prediction.
- We propose three methods for determining the mechanism of missingness using classification and clustering.

We found that the performance of the GraphSAGE model used for predicting sepsis was not greatly impacted by the different mechanisms, even at high missing rates. We hypothesize that the model may be missingness robust. We found that imputation with link prediction did not improve the results in the case of MCAR and MAR, however, in the case of NMAR it negatively impacted the performance. We discuss the implications and how this may be used as an alternative method for predicting NMAR. We found that using node2vec and clustering with DBSCAN and HDBSCAN showed promising results for determining MCAR as noise. However, results for determining the remaining two mechanisms were inconclusive. We found that using a GCN model to classify MCAR also showed promising results. However, it performed worse than our clustering strategy. Lastly, we attempted to cluster the negative embeddings created by the GCN model, however, while our clustering method indicated well-defined clusters, we were unable to further conclude on the results. Our method for simulating different types of missingness allows practitioners to make a more exhaustive analysis of how their model performs in the presence of missingness. Our three methods for determining missingness mechanisms allow practitioners to gain more insight into their data, and discover patterns of missingness.

Reuse of previous work

Parts of different sections in this work have been either partly or entirely reused from the pre-specialization, Handling Missing Data for Graph Machine Learning: A Review, Alexander Hansen, 2023 [1]. The following provides an overview of what has been reused:

- Mandatory Summary
  - Paragraph 1-2 has been reused from the introduction of Hansen 2023 [1], with some changes.
- Abstract
  - Sentence 1-3 has been entirely reused from the abstract of Hansen 2023 [1], with only a few changes.
- I. Introduction
  - Paragraph 1-2 has been entirely reused from the introduction of Hansen 2023 [1], with only a few changes.
- II. Related work
  - Paragraph 1 and 3 have been almost entirely reused from the related work of Hansen 2023 [1] with few modifications.
- III. Background
  - Subsection *A. Graph* has been entirely reused from combining subsections *A. Simple graph* and *B. Property Graph* from the Background of Hansen 2023 [1], with the addition of an explanation of a knowledge graph.
  - Subsection *B. Graph machine learning* has been entirely reused, without modifications from the subsection *C. Graph Machine learning* from Hansen 2023 [1].
  - The first paragraph of Subsection *D. Embedding*, has been entirely reused, from the subsection *D. Embedding*.
  - Subsection *E. Graph neural networks*, has been entirely reused, without modifications, from the subsection *E. Graph neural networks* from Hansen 2023 [1].
  - Figure 2 have been reused without any modifications, from Figure 2 of Hansen 2023 [1].
  - Figure 3a and 3b has been entirely reused without any modifications from figure 3a and 3b of Hansen 2023 [1].
  - Figure 4 has been entirely reused without any modifications from figure 4 of Hansen 2023 [1].
  - Subsection *F. Mechanisms of missingness*, has been entirely reused from *F. Mechanisms of Missingness* from Hansen 2023 [1] without any modifications.
  - Subsection G. Imputation, has been entirely reused, without modifications, from the subsection G. Imputation from Hansen 2023 [1].
- IV. Simulating missingness mechanisms
  - In Subsection *B. Evaluation*, the brief explanation of the MIMIC-IV dataset has been entirely reused from Hansen 2023 [1] Section *V. Setup for Evaluating Method of Handling Missingness*

Determining and Handling Missing Data for Graph Machine Learning

Alexander Højgaard Hansen
Department of Computer Science
Aalborg University
Aalborg, Denmark
ahha19@student.aau.dk

Freja Herreborg Pedersen
Department of Computer Science
Aalborg University
Aalborg, Denmark
fhpe19@student.aau.dk

***Abstract*—In the field of machine learning, learning on graphs has become increasingly popular due to its usefulness in modeling data sets with complex relationships and dependencies. However, missing values in graph data pose a challenge for machine learning. The way missing values are handled has significant implications, as it can inhibit the performance of models by introducing bias and lessening the generalizability. The impact of how different missingness mechanisms affect downstream tasks for graphs is not a well-studied subject. In this work, we simulate missingness in a graph of medical data from patients and use the tasks of sepsis prediction with a graph convolutional network (GCN) model to evaluate. To explore if the missingness can be handled, we formulate link prediction as an imputation method and evaluate if it can improve the models' prediction performance despite missingness. Other work has been done to show how Explainable Boosting Machines (EBM) can help gain insights into missingness mechanisms, and propose how EBM may be used to test for MCAR. However, they present these methods while assuming a tabular data structure. In this work, we examine how the type of missingness may be determined by using techniques such as classification and clustering.**

***Index Terms*—Graph machine learning, Property graphs, Knowledge graphs, Embedding, Imputation, Missingness mechanisms, Clustering, Classification**

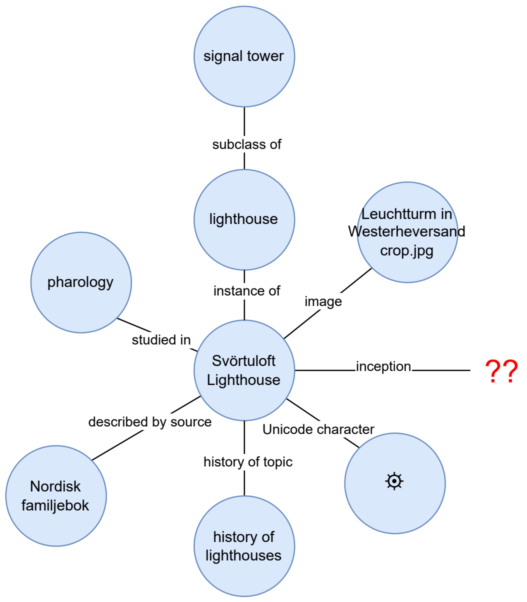


Fig. 1: Svörtuloft Lighthouse is used as an example entity in a graph where the `inception` is missing.

I. Introduction

Data is all around us. Every year, millions of petabytes of data are recorded [2]. Machine learning has become an increasingly popular approach to learning from this data, where many of these models and methods assume a tabular data structure. However, a graph-based representation has advantages over a tabular representation, as it can more naturally model data sets with complex relationships and dependencies [3]. To illustrate these points, consider a citation network. Representing this in a tabular structure may not be as simple, as a paper may cite or be cited by a varying number of papers. Furthermore, these papers may have a number of properties and a set of papers they relate to. This makes it challenging to represent with a single row of a tabular-based structure, as a fixed number of columns cannot be used to represent this. A graph structure more naturally accommodates this relationship. Graphs are used in a variety of domains, for example, IoT networks of sensors [4], traffic networks [5], and knowledge graphs which are graph representations of knowledge [6]. To illustrate, a simple example of a graph can be seen in Figure 1.

The use of graph neural network (GNN) models has increasingly become popular for machine learning tasks, such as node classification [7] and relation prediction [8]. Models such as the graph convolutional network (GCN) and graph attention network (GAT) have seen ground-breaking results in the field of deep learning [9]. Unlike traditional neural networks, these models assume a graph-based representation of the data and can leverage the inherent structural information available in them. GNN models are powerful, but missing values pose a challenge for these models. Missing values may be viewed as a type of erroneous data, and thus warrants detection and correction measures. Data may be missing for various reasons that may occur during recording or when merging data sets [10]. Samples may be missing due to a sensor malfunction during recording. Participants filling in a questionnaire may decide to leave answers blank. Physicians recording medical tests for patients may decide to leave the results unrecorded if they are not abnormal [11]. As such, missing values are an unavoidable fact of life when dealing with data sets. When modeling the data as graphs, the missingness will manifest as missing nodes, edges, or attributes. Dealing with the missingness problem is important, as it can otherwise greatly impact a model's performance by introducing bias and making it less generalizable [12]. Depending on the underlying reason, missing data can be categorized into three different missingness mechanisms: missing completely at random (MCAR), missing at random (MAR) and not missing at random (NMAR) [13]. The underlying reason for the missingness is valuable information to researchers, as it can provide important insights into the data and the data collection process.

Previous work has suggested methods to classify mechanisms of missingness; for example, Gad-Elrab et al. [14] present a method to test for MCAR using Explainable Boosting Machines (EBM). However, this method assumes a tabular dataset. In this work, we examine how the mechanism of missingness may be determined using clustering and classification using a sub-graph of Wikidata. The impact of the different missingness mechanisms for downstream machine learning tasks is not a well-studied subject for graphs [1]. While previous research has examined how to handle different missingness mechanisms for tabular data, the same has not been done for graph data [15]. In this work, we simulate missingness based on missingness mechanisms and formulate link prediction as an imputation method. We examine how the downstream task of sepsis prediction, in the MIMIC-IV dataset, is impacted by missingness, and how performance may change with imputation.

Our contribution can, therefore, be summarized as follows.

- We propose a method for simulating missingness mechanisms in a graph, given a specified rate.
- We empirically evaluate how the different mechanisms and missing rates affect downstream tasks for graph machine learning, specifically sepsis prediction.
- We formulate link prediction as an imputation method for missing edges in a graph and investigate if it can improve the downstream task of sepsis prediction.
- We propose three methods for determining the mechanism of missingness using classification and clustering.

The rest of the paper is structured as follows: In Section II we review related work. In Section III we provide background and preliminary definitions. Our main contributions are presented in Sections IV, V and VI, where we propose and evaluate methods for determining the impact of missingness mechanisms, imputing edges, and determining missingness mechanisms respectively. We discuss the results of the three evaluations in Section VII and conclude in Section Section VIII.

II. Related Work

Chami et al. [16] describes a taxonomy of graph representation learning, presenting and generalizing popular algorithms for semi-supervised and unsupervised learning of graph representations into a single, consistent approach. The taxonomy encompasses topics such as network embedding and graph neural networks, including models such as GNN, GCN, and algorithms such as DeepWalk and node2vec. While the authors discuss link prediction as a method for predicting missing or unobserved links, they do not specify how this may be used to handle missingness for downstream tasks on graphs. In this work, we examine how link prediction, formulated as an imputation technique, may be used to improve the downstream task of sepsis prediction.

Gad-Elrab et al. [14] present how recent advances in Explainable Boosting Machines (EBM) can help gain insights on missingness mechanisms. They present how EBM may be used to detect or even alleviate risks introduced in models by imputation algorithms. They also propose how EBM may be used to test for MCAR. However, they present these methods while assuming a tabular data structure. In this work, we present how missingness can be handled in graphs using imputation and how the mechanism of missingness may be determined using clustering and GNN classification.

Lin and Tsai [15] review a decade of imputation technique research. The authors describe statistical techniques such as expectation maximization (EM), gaussian mixture model (GMM) and, multiple imputation by chained equations (MICE) as well as machine learning-based techniques that use models such as K-nearest neighbors (KNN), multilayer perceptron (MLP), and random forest (RF). While the authors present these techniques while also considering missingness mechanisms, all the techniques assume a tabular representation of the data and do not leverage the structural information inherent to graph-based representations. In this work, we present and explore an imputation method for graphs to mitigate edge missingness.

III. Background

In this section, we introduce the background necessary for understanding our findings.

A. Graph

Simple graphs can be represented as:

$$G = (V, E)$$

where G represents the graph itself, V is the set of nodes or vertices, and E is the set of edges [17, p. 148].

- Vertices or nodes: Nodes represent entities in a system. In a social network, nodes may be people. In an IoT network, a node may be an IoT device or a sensor.
- Edges: Edges define relationships between nodes or entities. Edges can be used to model interactions, dependencies, or other connections between nodes. In a social network, an edge could denote friendship between two people. An edge may be directed or undirected. An undirected edge indicates a symmetric relationship between nodes, and a directed edge indicates an asymmetric one.

A graph may be either homogeneous or heterogeneous. For example, in a homogeneous social network, all nodes consist of people, and all edges denote a relationship between people; the node type and edges are the same.

Machine learning on graphs often involves using a property graph, which extends the simple graph with labels and properties, allowing for heterogeneous graphs. It allows for modeling more complex relations and more complex systems. A property graph [18] extends the simple graph and can be represented as:

$$G = (N, R, \iota, \lambda, \tau)$$

where N is a set of nodes. R is a set of relationships. ι is a function that maps a node or relationship and a property key to a value. λ is a function that maps nodes to a set of labels. τ is a function that maps relationships to a relationship type. To illustrate, an example of a property graph can be seen in Figure 2 showing a social network. We show how it can be represented as $G = (N, R, \iota, \lambda, \tau)$.

- N = $\{n1...n10\}$
- R = $\{r1...r9\}$
- ι: $\iota(n1, name) \rightarrow Ben$, $\iota(n2, name) \rightarrow James$, $\iota(n3, name) \rightarrow Sasha$ ... $\iota(n9, Location) \rightarrow Copenhagen$
- λ: $\lambda(n1) \rightarrow Dancer$, ..., $\lambda(n4) \rightarrow Artist$, ..., $\lambda(n7) \rightarrow ArtStudio$, ...
- τ: $\tau(r1) \rightarrow friend$, ..., $\tau(r4) \rightarrow interacted$, ..., $\tau(r8) \rightarrow frequents$, $\tau(r9) \rightarrow collaborate$, $\tau(r10) \rightarrow collaborate$

An extension of the property graph is the knowledge graph, where nodes and edges are enriched with schemata, rules, and ontologies. This helps to validate, structure, and define the semantics of the graph while enabling a machine to reason over the graph [19, p. 45].

B. *Graph machine learning*

Machine learning is an essential tool for deriving patterns and insights from graphs. It has a wide range of uses, such as forecasting traffic [5] or analyzing the structure of proteins [6]. Machine learning may be either supervised or unsupervised depending on the model or tasks. In supervised learning, the data set will have labels. Depending on the tasks, labels may be for the whole graph, sub-graphs, nodes, or edges, and the model will try to learn the mapping between a given component and its label by training. Examples of supervised learning include prediction [20] and classification [21]. In unsupervised learning, the data set is not labeled. Instead, the model will try to learn underlying patterns in the data. Examples of unsupervised learning include clustering [22] and embedding [23]. In this work, we use the following tasks:

- **Prediction**: Given a graph G and a target variable V, the goal of prediction is to define a function that maps a component of the graph to the variable [8]. An example of prediction is link prediction, where the goal is to predict missing links between pairs of entities [24].
- **Classification**: Classification is a type of prediction task where the goal is to determine the function f, such that given a graph G, the function will correctly map the graph or its components to a set of classes C [25, p. 10]. An example could be node classification, to determine whether a person in a social network is a celebrity or not. The aforementioned example is a case of binary classification, as there are only two classes in C.
- **Clustering**: The goal of clustering is to determine the function f, such that, given a graph G, the function will group the graph or its components into clusters based on similarity [25, p. 8-9].

C. *Clustering algorithms*

A clustering algorithm is an algorithm that groups a graph, or its components, into clusters based on similarity. In this work, we use the clustering algorithms K-means [26], DBSCAN [27] and HDBSCAN [28]. K-means is a partitioning-based clustering method that divides the dataset into K non-overlapping clusters. Because of how clusters are found, K-means assumes that clusters are convex-shaped. This allows for an efficient algorithm [29], however, it is less suitable than a density-based algorithm, when the shape of the dataset is arbitrary [30]. DBSCAN [27] is a density-based clustering method that groups data points based on density. How clusters are formed can be parameterized by ϵ and a minimum samples parameter. If samples do not satisfy the conditions of the parameters, they will be labeled as noise. ϵ specifies the maximum distance for a sample to be considered in the neighborhood of another sample. Minimum samples specify how many samples in a neighborhood are required to form a cluster. A sample is a core sample if there are at least minimum samples within an ϵ distance of the sample, including itself. When a cluster contains less than the minimum samples, it is denoted as noise. This is also the case if a sample is not within ϵ of a core sample and is not a core sample itself. Together, ϵ and minimum samples affect how many samples are labeled as noise and how sparse or dense the clusters will be. Clusters by DBSCAN can be any shape. HDBSCAN [28] extends DBSCAN by using hierarchical density-based clustering. The density-based cluster hierarchy allows the clusters to be of varying densities, which is not possible when using DBSCAN. ϵ is not specified, instead, the algorithm determines the different density levels of the clusters.

D. *Embedding*

Many downstream tasks, such as node classification and link prediction on graphs, rely on node and graph embedding, as many machine learning algorithms for graphs learn on numerical vectors, i.e., vectors of real numbers. In order to represent a graph as a vector, embedding can be used. Embedding is a way of mapping a complex data structure of high dimensionality, such as a graph, into a fixed-length vector that captures key features while reducing the dimensionality [16]. For graph embedding, these key features capture the relational information within a graph. For node embedding, the embedding captures the node's local relationships within the graph. A toy example of node embedding can be seen in Figure 3, where the graph is embedded in 2D vector space. Figure 3a shows the graph used for the embedding examples. Figure 3b shows node embeddings, where nodes in three

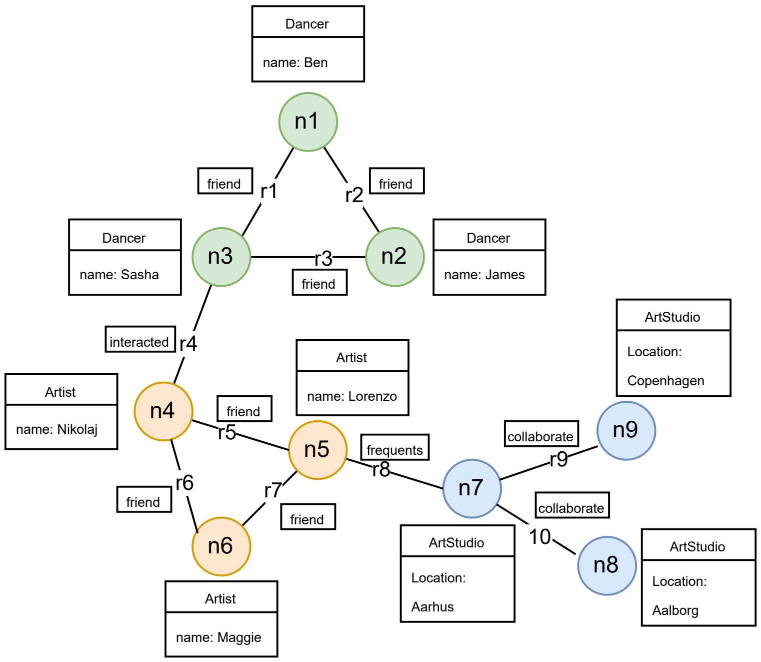


Fig. 2: An example of a property graph showing a network of dancers, artists, and art studios.

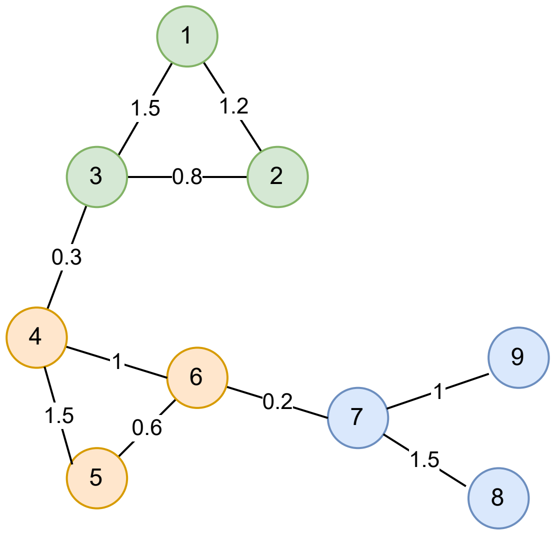
neighborhoods green, orange, and blue are also close together in the embedding space.

To represent the local neighborhoods, embedding techniques can involve methods for sampling neighborhoods, such as random walk [31] or message passing [9]. An example of an embedding method is the node2vec algorithm [31], which computes embeddings for nodes in a graph. It works by sampling a fixed number of random walks on the graph. The result will be a fixed number of sequences of nodes. These sequences then get passed to a skip-gram model, which computes the embedding for each sequence. The skip-gram model is a language model designed to maximize the co-occurrence probability of words that appear in a context window [32]. In the case of graphs, the words are nodes, and the context window is the sequence of nodes generated from the random walk. The random walk strategy can also be parameterized to be biased by variables p and q which guide the walk. The two parameters allow the search procedure to balance between a depth first search (DFS) and breadth first search (BFS) strategy [31].

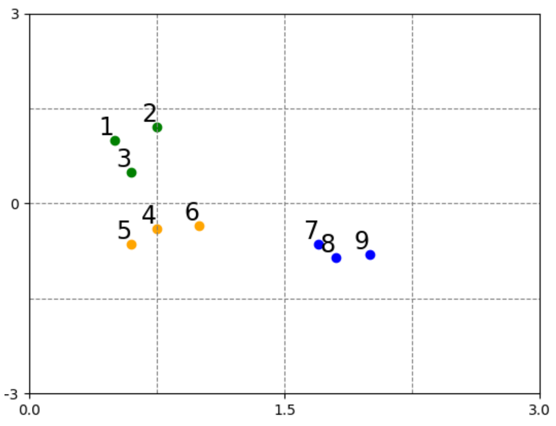
E. Graph neural networks

A graph neural network (GNN) is a type of supervised machine learning algorithm that can be applied to graphs [9]. The model is able to leverage the structural information inherent in graphs and can represent key information about nodes and local contexts. The key concept of the GNN is the message passing algorithm, as it is the method by which it learns graph representations. To illustrate, Figure 4 shows two iterations of the message-passing algorithm with node 4 as the target. In the first iteration, every node will send out a message to its neighbors. This means each node will have several incoming messages, which it will aggregate using an aggregation function, where the implementation will depend on the model. As such, each of node 4's neighbors aggregates its corresponding neighbors. In the second iteration, every node will have an updated representation based on its neighbors. Finally, node 4's neighbors 3, 5, and 6 are aggregated to update node 4's representation. This process is performed for every node, and the number of iterations will dictate how far information gets passed in the network.

An example of a GNN is the graph convolutional network (GCN) model [33], which extends the message passing algorithm with the concept of convolutional layers, commonly used in image processing. In the message-passing phase, each node will send out a message consisting of its features as a vector. In the aggregation, firstly a simple order-agnostic aggregation function like mean, max, and sum is applied to aggregate every incoming feature vector. The aggregated feature vector is then passed through a dense neural network layer, which transforms the feature vector. Each layer gets passed the previous aggregated and transformed feature vector. In the case of the GCN, the number of iterations will depend on the number of layers in the dense neural network [34].



(a) Input Graph G



(b) Node Embedding

Fig. 3: A small example of what node embedding can look like, when nodes are embedded to 2D-vectors. [23].

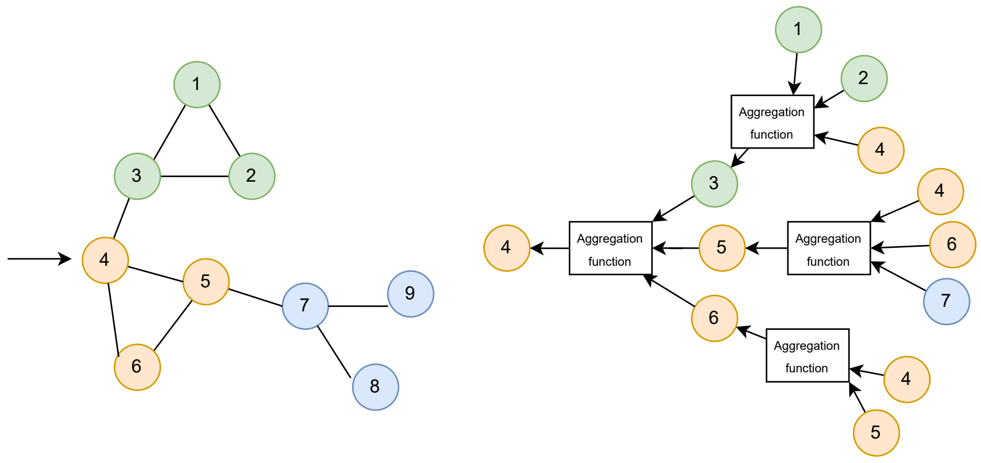


Fig. 4: An example of the message passing algorithm, showing two iterations where the target node is node 4.

F. Mechanisms of missingness

Data points can be missing for a variety of reasons. Missingness may be due to observed or unobserved variables and can be described using missingness mechanisms. Missing data can be categorized into the following categories: missing completely at random (MCAR), missing at random (MAR) and not missing at random (NMAR) [11].

- Data is MCAR when the probability of missingness is unrelated to the value of the data point, observed, or unobserved variables. As such, the probability of a data point missing is the same as for any other data point [12]. To illustrate, consider a clinical trial, where patients are asked to record different information at certain time intervals. Due to a malfunction in the data collection system, random samples went missing. In this case, missingness is not correlated to any recorded or unrecorded variables and therefore MCAR.
- Data is MAR when the probability of missingness for a data point is determined from an observed variable [12]. To illustrate, in the same clinical trial, patient were asked to record their pain level at different time intervals. Some patients forgot to perform the recordings. The missingness was not completely random, as it could likely be inferred from information about their health condition. Clinicians found that patients in good health were more likely to forget to record pain levels than patients in poor health. In this case, the missingness is correlated to the observed variable ”health condition” and therefore MAR.
- Data is NMAR when the probability of missingness is related to an unobserved variable [12]. To illustrate, in the same clinical trial, patients were asked to record the side effects of their anti-depressant medication. Some participants who experienced side effects were less likely to record their side effects, as they were afraid of getting

switched to a different medication. In this case, missingness is correlated to the unobserved variable "motivation".

G. Imputation

One method for handling missing data is imputation [35] [36]. Imputation involves substituting missing values with some new value. A simple and common approach is to impute the missing values with the mean value for the given variable, i.e., mean imputation. A more sophisticated approach is multiple imputation. For multiple imputation, when substituting a value, multiple variables will be considered [37]. Another method is creating a prediction model and training it with all features in the data set except for the feature to impute. This model can then be used to predict the missing value, thus imputing it. When imputing data in graphs, the structure of a graph can be leveraged by the imputation methods [38].

IV. Simulating missingness mechanisms

In this section, we present our method for simulating the different missingness mechanisms in a property graph (§ IV-A). We then demonstrate its utility by examining the impact of different types of missingness on a downstream graph-based machine learning task (§ IV-B). The results are discussed in Subsection IV-C

A. Proposed method

We introduce missingness to the graph by removing edges from the graph based on a missing rate and mechanism. For MCAR, we uniformly remove edges at random. The same method will be used when removing edges for the remaining mechanisms. For MAR, we remove edges randomly based on an observable conditional variable, where the variable will be a node attribute. For NMAR, the missingness depends on an unobservable conditional variable, which will also depend on a node attribute. The attribute used is removed from the graph to make the variable unobservable.

B. Evaluation

We evaluate the performance impact of different types of missingness for downstream tasks while applying our method for simulating missingness on the Medical Information Mart for Intensive Care IV (MIMIC-IV) dataset [39] and the task of sepsis prediction. The MIMIC-IV data set contains healthcare data of real-world patients recorded from hospitals in Boston, Massachusetts, USA. This is hospital-level data and data from the ICU and emergency department, including data about patients, their medication, admissions, diagnoses, and more. We base our implementation on Hansen et al. [21], where the MIMIC-IV data set will be transformed into a bidirectional heterogeneous graph. Domain hierarchies are used for embedding initialization. The number of nodes can be seen in Table I and the number of relations in Table II.

We use the missing rates 10%, 20%, 30%, 50%, 70%, 90%, 99.99%. For MCAR we use NumPy's uniform [40] function to remove edges. For MAR, we choose the conditional variable to be age. As such, patients between the ages of 17 and 54 are the only patients where their edges may be removed. For NMAR, we choose the conditional variable to be sex, where male patients are the only patients where edges may be removed. Both MAR and NMAR variables are chosen based on Getzen et al. [41], where both patients between 17-54 and male patients are part of a "data-lacking group" since these groups are statistically less likely to seek or have access to care. Because of the definitions we have used for the missingness mechanisms, the actual missing rate for each mechanism will not be identical to the specified rate. This is illustrated in Table III, where for each edge type, the proportion of edges that are actually removed, based on the specified rate, is denoted. The actual missing rates for MAR and NMAR are lower, as the missingness can only be simulated for the edges based on the conditional variable used, as opposed to all edges for MCAR.

Node type	Count
Patient	128,605
Medication	1,749
LabTest	664
Procedure	1,228
Total	132,246

TABLE I: Node types and count of each in the MIMIC-IV graph.

Relation type	Count
Patient-LabTest	10,548,163
Patient-Medication	3,926,669
Patient-Procedure	194,836
Total	14,669,668

TABLE II: Relation types and count of each in the MIMIC-IV graph.

Edge type	MCAR	MAR	NMAR
Patient-Labtest	1	0.38	0.48
Patient-Medication	1	0.33	0.5
Patient-Procedure	1	0.36	0.51

TABLE III: The actual number of edges that are removed based on the specified missing rate for each mechanism and edge type.

To evaluate the impact of missingness, we use the task of sepsis prediction, meaning whether a patient will be diagnosed with sepsis based on information about the patient and their stay in care. Predictions will be done using the GraphSAGE model [42] created with the Python framework PyTorch [43]. The best hyperparameters found in Hansen et al. [21] will be used. To evaluate the performance of sepsis prediction, we use the metric F1 score, which can be derived from the Fβ score. The formula for the Fβ score can be seen below:

$$F\beta\ Score = \frac{(1+\beta^2)\cdot precision \cdot recall}{\beta^2 \cdot precision + recall}$$

The β parameter represents the ratio of recall importance: F1, where β=1 would give the harmonic mean of precision and recall.

Recall and precision are defined below:

$$recall = \frac{\#True\ Positives}{\#True\ Positives + \#False\ Negatives}$$

$$precision = \frac{\#True\ Positives}{\#True\ Positives + \#False\ Positives}$$

Recall measures how many positive samples the model correctly guesses. Precision measures of how many of the positive guesses are correct.

C. Results

In Figure 5 we show the results of how each missingness mechanism affects the F1 score of sepsis prediction. Figure 5a shows how the mechanisms, simulated across all edge types, impact the F1 score of sepsis prediction. Figure 5b shows the F1 score of when simulating for the Patient-LabTest edge type, Figure 5c for the Patient-Medication type and Figure 5d for the Patient-Procedure type. Except for the Patient-Procedure edge type, we generally find that MCAR impacts the F1 score the most, resulting in the lowest performance for each missing rate. This is not so surprising, as MCAR is the mechanism that exhibits the highest actual missing rate. However, for the Patient-Medication and Patient-Procedure type, we find that NMAR impacts the performance very similarly to MCAR, despite exhibiting roughly half the actual missing rate. When simulating for all edge types, we find that NMAR impacted the performance more than MAR for missing rates higher than 55%. Overall, we find that the performance of the model was not significantly impacted when only simulating missingness for a single edge type, even at high rates. For this reason, we only consider missingness across all edge types for the following experiment, as it was the case that had the largest impact on performance.

V. Edge imputation

In this section, we present our method for imputing missing edges in a graph (§ V-A). We demonstrate its use by enriching a property graph with new imputed edges and evaluate the impact on a downstream graph-based machine learning task (§ V-B). The results are discussed in Subsection V-C.

A. Proposed method

We propose a method for the imputation of edges in a graph by proposing link prediction as an imputation method and investigating if it can improve the performance of downstream tasks. After simulating missingness, we perform link prediction to enrich the MIMIC-IV graph by scoring potentially missing edges and selecting the top K edges to add to the graph, where K is equal to the number of removed edges. We use the models ConvE and TuckER to perform link prediction. The ConvE model is presented by Dettmer et al. [20] where they show that the model is effective at modeling nodes with a high node indegree, which is very common in complex knowledge graphs. The model is a multi-layer convolutional network model for computing embeddings of knowledge graphs, which can also be used for link prediction.

Parameter	**Range**	**Best found**
Input dropout	{0.0, 0.1..0.5}	0.0
Feature map dropout	{0.0, 0.1..0.5}	0.1
Projection layer dropout	{0.0, 0.1..0.5}	0.5
Embedding dimensionality	{100, 200}	100
Batch size	{64, 128, 256}	256
Learning rate	{0.001, 0.003}	0.003

TABLE IV: Parameters for ConvE hyperparameter tuning.

Parameter	**Range**	**Best found**
Embedding dimensionality	{16, 32, 48..256}	224
Dropout values	{0.1, 0.2..0.5}	(0.5,0.5,0.1)
Learning rate	{0.01, 0.005, 0.003, 0.001, 0.0005}	0.003

TABLE V: Parameters for TuckER hyperparameter tuning.

Parameter	**Range**	**Best found**
Hidden dimensions	{32, 64, 128, 256}	64
Model dept	{2..12}	11
Learning rate	{0.0005, 0.001, 0.005, 0.01}	0.005
Dropout rate	$U(0.0..0.5)$	0.44

TABLE VI: Parameters for GCN hyperparameter tuning.

In Balažević et al. [44] they present TuckER, a linear model based on Tucker decomposition of the binary tensor representation of knowledge graph triples. The model can compute embeddings of knowledge graphs and perform link prediction.

B. Evaluation

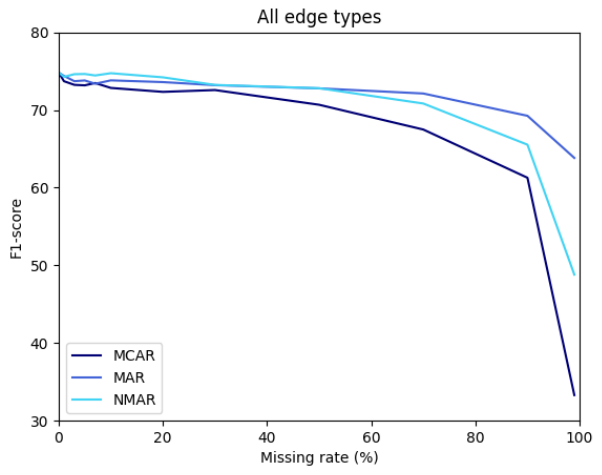
We implement the models using the PyKEEN (Python KnowlEdge EmbeddiNgs) library [45] and tune the models according to the parameter ranges presented in their original work. We use grid search over the parameters for a subset of all triples from the MIMIC-IV graph. The subset consists of ≈15,000 triples, ≈5000 of which are used as validation. For ConvE, hyperparameter ranges can be seen in Table IV. The Adam optimizer and early stopping are also used. For TuckER, the hyperparameters can be seen in Table V. Batch normalization and the Adam optimizer with early stopping are used. We evaluate the performance of the imputation based on the F1 score for the sepsis prediction task.

C. Results

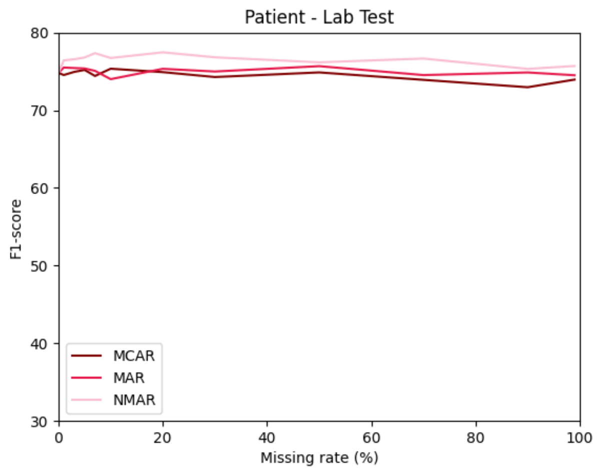
Figure 6a shows the F1 score when TuckER and ConvE have been used as preprocessing to enrich the graph with imputed edges. The baseline shows the F1 score without any imputation. Figure 6b shows the same but for the MAR mechanism, and Figure 6c shows for NMAR. What we find, in general, is that using link prediction as imputation does not improve the F1 score of the downstream task. When the mechanism is MCAR and MAR, the F1 score does not deviate much from the baseline. What is interesting, however, is when the mechanism is NMAR, there is a decrease in performance.

VI. Determining missingness mechnanism

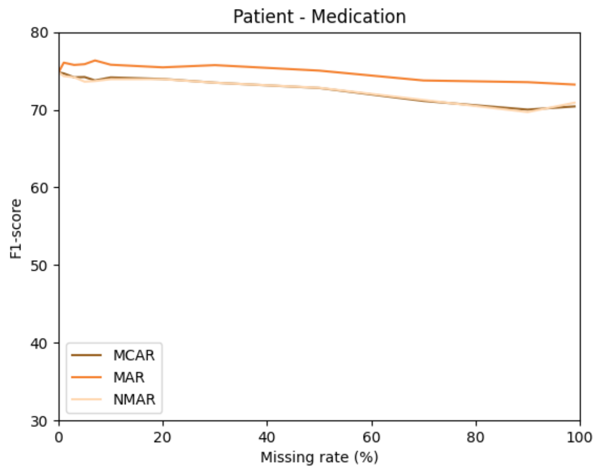
In this section, we propose our three methods for determining the mechanism of missingness in a graph (§ VI-A). We examine the utility of the three methods by evaluating



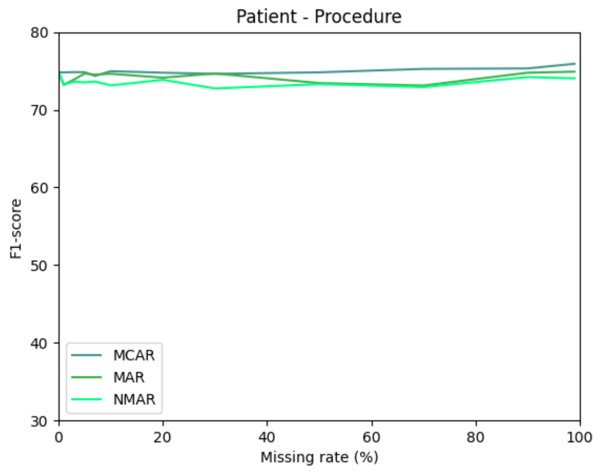
(a) Missingness simulated for alle edge types



(b) Missingness simulated for the patient to lab test edge type



(c) Missingness simulated for the patient to medication edge type



(d) Missingness simulated for the patient to procedure edge type

Fig. 5: Graphs showing how the task of sepsis prediction is impacted by the simulations of the different missingness mechanisms across the different types of edges in the graph.

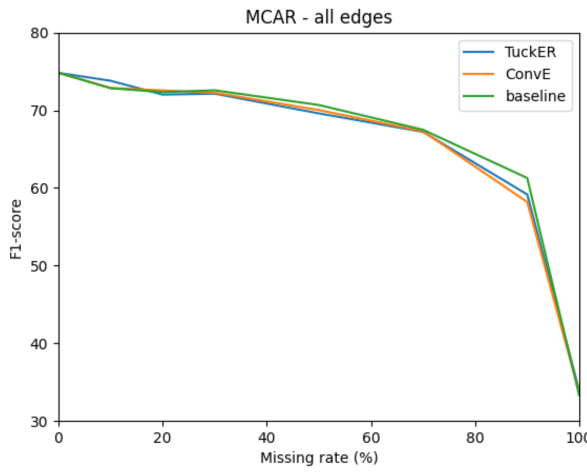
whether the methods can determine the missingness mechanisms present in a knowledge graph (§VI-B). The results are discussed in Subsection VI-C.

A. *Proposed method*

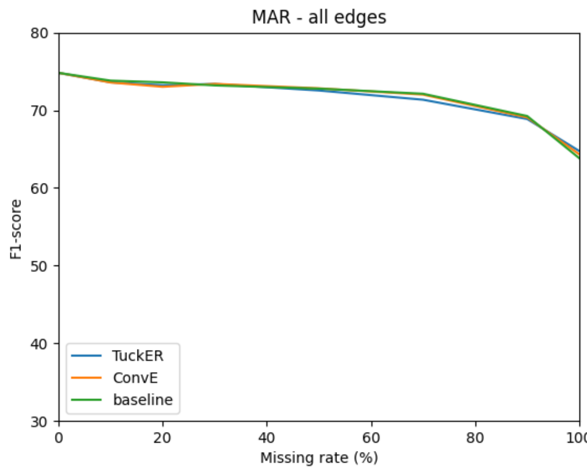
For the first method, we propose embedding the nodes of the graph using node2vec and using clustering to determine the missingness mechanism. We use K-means, DBSCAN, and HDBSCAN to hopefully discover clusters of MCAR, MAR, and NMAR. For the second method, we propose using a GCN model to classify MCAR. Lastly, we propose using the embeddings created by a GCN from the last hidden layer, after performing MCAR classification and determining the missingness mechanism using clustering. We use the embeddings predicted not to be MCAR, and perform clustering on them to hopefully identify clusters exhibiting MAR and NMAR. For this, we use the same aforementioned clustering algorithms.

B. *Evaluation*

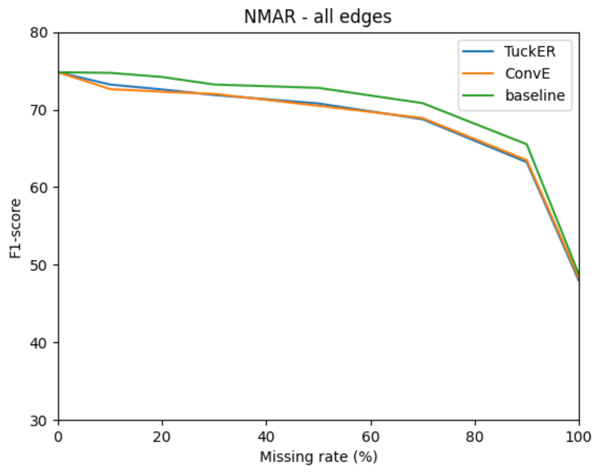
To explore how missingness mechanisms can be determined, we use a sub-graph of Wikidata. Wikidata is a knowledge graph used to store information from Wikimedia projects, including Wikipedia [46]. Entries in Wikimedia are written by different users, with over 24,000 users active in the last 30 days [47]. This means that different patterns may emerge for similar entities, as information is primarily added by humans. For example, if one user denotes the date a city was founded with the relation `start time (P580)` instead of the correct relation `inception (P571)`, then two cites may have the same date information, but with different semantics. To examine whether it is possible to determine the underlying missingness mechanism, we construct an undirected graph consisting of locations in Iceland as nodes, where all nodes are missing the relation `inception`. Each location node only has its immediate relations, except for a type hierar-



(a) The F1-score of the sepsis diagnosis for each missing rate using the missingness mechanism MCAR.



(b) The F1-score of the sepsis diagnosis for each missing rate using the missingness mechanism MAR.



(c) The F1-score of the sepsis diagnosis for each missing rate using the missingness mechanism NMAR.

Fig. 6: Graphs showing the f1-score performance of sepsis diagnosis, when Tucker and ConvE are used as preprocessing to impute missing edges, given the three types of missingness mechanisms

chy. The type hierarchy consists of four layers, meaning that for every `<location isInstance o1>`, we also retrieve `<o1 isSubclassOf o2>`, `<o2 isSubclassOf o3>` and `<o3 isSubclassOf o4>`. The graph is created using the deep graph learning (DGL) library. The graph consists of 500 nodes, where 100 of the locations are simulated as MCAR by randomly selecting 100 locations from the set of locations that do have an inception date, and removing their inception date relation. However, we do not know the actual distribution of labels for the remaining 400 locations. Therefore, there may be more than 100 MCAR samples. We use K-means as it is a very efficient and scalable clustering algorithm, which provides distinct and non-overlapping clusters. DBSCAN was chosen as it can identify clusters of arbitrary shapes and has been acknowledged as an algorithm that has stood the test of time [48]. HDBSCAN was chosen, as it addresses some of the limitations of DBSCAN, by being able to detect clusters of varying densities.

For node2vec, we use the implementation from Cohen [49], and we choose the parameters seen in Table VII. The implementation of the clustering algorithms is from Pedregosa et al. [50], and we tune the clustering algorithms using grid search and use the silhouette score to determine the best parameters. The ranges and parameters tuned can be seen in table Table VIII, Table IX, and Table X. We train the GCN model in a supervised setting, using a separate graph, which also consists of 500 total locations, 100 of which are MCAR. We tune the GCN model by performing 300 iterations of tree-based Parzen estimation for hyperparameter tuning, shown in Table VI. Adam optimizer and early stopping are also used.

Parameter	value
Embedding dimensionality	128
walk length	80
Number of walks	10
p	1
q	1

TABLE VII: parameters for node2vec.

Parameter	Range	Best found
n_clusters	{2..30}	2
n_init	{10,20,30}	10

TABLE VIII: parameters for KMeans hyperparameter tuning.

Parameter	Range	Best found
ϵ	{0.1,0.5,1,2..5}	3
min_samples	{2..10,15,20,30}	5
metric	{Euclidean, Manhattan}	Euclidean

TABLE IX: parameters for DBSCAN hyperparameter tuning.

Parameter	Range	Best found
min_samples	{1..10}	1
min_cluster_size	{2..10}	8
metric	{Euclidean, Manhattan}	Euclidean
α	{0.1,0.5,1,2,3}	3
cluster_selection_epsilon	{0.1,0.2..1.0}	1.0

TABLE X: Parameters for HDBSCAN hyperparameter tuning.

We use binary cross entropy as the loss function.

To evaluate the classification task, we use the F5 score and F1 score derived from the Fβ score. We choose to value recall higher than precision since we don't want any MCAR in our remaining samples after the classification, as we would like to derive useful clusters and the underlying patterns that occur for NMAR and MAR. We also know that some unlabeled samples could be MCAR, and we do not want to punish the model for potentially classifying these. For this reason, we value the F5 score more than the F1 score when evaluating our results.

To evaluate our clusters we use the silhouette score [51] as a measure of cluster goodness. The score is defined as follows:

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$$

where $s(i)$ is the silhouette score for a single data point i, $a(i)$ is the intra-cluster distance for point i, $b(i)$ is the inter-cluster distance for point i. $max(a(i), b(i))$ is used to normalize the score between -1 and 1. The silhouette score is computed for every data point, and the mean is computed to get a single score. A silhouette score of 1 indicates that the clusters are well apart from each other and distinguishable. A score of -1 indicates misclassification and a score of 0 indicates overlapping clusters or that the clusters are indistinct.

C. Results

Figure 7a shows a principal component analysis (PCA) transformation of the vectors created by node2vec, where each dot is the vector of a location. Orange-colored dots are vectors labeled MCAR while the remaining blue are unknown. Figure 7b, Figure 7c, and Figure 7d show the clusters found using the K-means, DBSCAN and HDBSCAN algorithms respectively. K-means obtains a silhouette score of 0.043, DBSCAN a score of 0.084, and HDBSCAN a score of 0.136.

Algorithm	F5 score	F1 score	recall	precision
node2vec + DBSCAN	0.92	0.76	0.94	0.64
node2vec + HDBSCAN	0.94	0.73	0.96	0.59
GCN	0.83±0.07	0.60±0.05	0.85±0.08	0.47±0.07

TABLE XI: Metrics for classifying MCAR. The scores for DBSCAN and HDBSCAN are for classifying MCAR as noise using node2vec embeddings.

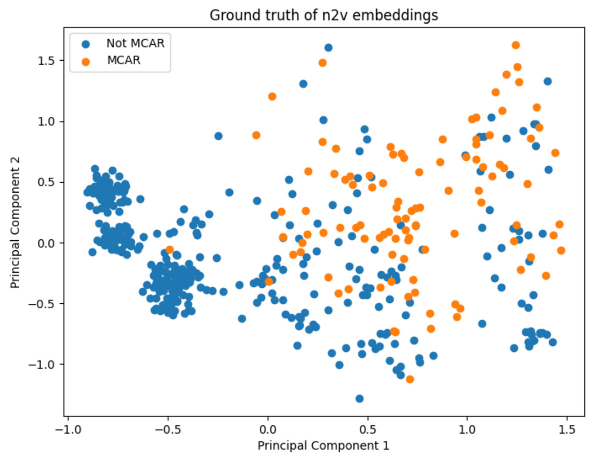
While the silhouette score indicates that the clusters are not distinct nor clearly separated, we also evaluate how well the samples determined as noise by DBSCAN and HDBSCAN corresponded to MCAR. The F5 score, F1 score, recall, and precision can be seen in Table XI. For the F5 score, we find that both DBSCAN and HDBSCAN perform well with an F5 score of 0.92 and 0.94, respectively. The algorithms can correctly classify the majority of samples labeled as MCAR. As such, while we find that DBSCAN and HDBSCAN perform well at classifying MCAR, the silhouette scores for the remaining clusters are quite low.

Using our GCN model to classify MCAR locations in our Wikidata graph, we obtain the average scores on the test set across five separate runs as seen in Table XI. The table shows the F5 score, F1 score, recall, and precision. The model obtains an average F5 score of 0.83. We verify the statistical significance of the model's performance by using a t-test, with 10 separate runs split into two groups. We obtain a P value of 0.91 and a 95% confidence interval. We can, therefore, conclude that the difference in results between runs is not statistically significant.

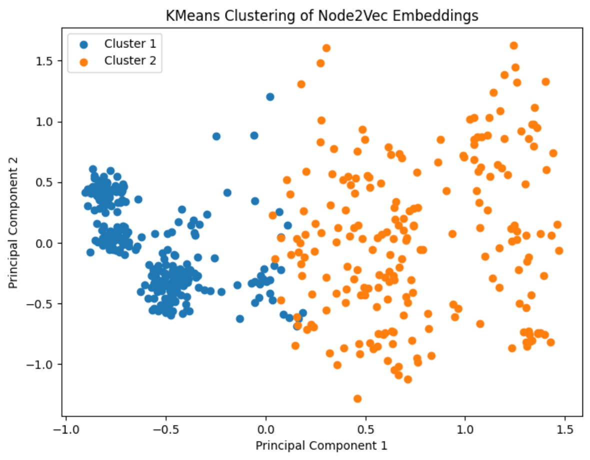
After performing MCAR classification using the GCN model, we performed clustering on the embeddings predicted as not MCAR using DBSCAN, HDBSCAN, and K-means. The clusters are visualized in Figure 8, where Figure 8c shows resulting clusters for DBSCAN, Figure 8d for HDBSCAN and Figure 8b for K-means. Figure 8a shows the actual distribution of the labels. The figures also show the embeddings that were predicted as MCAR by the GCN model. Clustering on the negative embeddings resulted in the following silhouette scores of 0.76, 0.44, and 0.59 for DBSCAN, HDBSCAN, and K-means, respectively. We find that DBSCAN obtains the highest silhouette score, with a score that indicates clear and well-defined clusters. However, we are not able to determine whether the clusters specifically correlate with NMAR or MAR, making the results inconclusive.

VII. Discussion

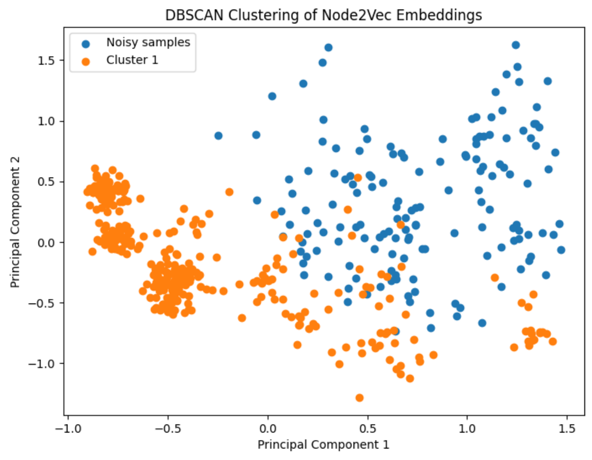
We find that the classification performance was not impacted as much as we expected when performing sepsis prediction, even at high missing rates. This could indicate that the GraphSAGE model is missingness robust and potentially be an ideal model to use when datasets have large amounts of missing relations. We found that when data is NMAR and link prediction is applied as imputation, the classification performance decreases. It is interesting as it is also what is found to be the case in literature for imputation in tabular datasets [14]. As such, future research may avoid using imputation, when



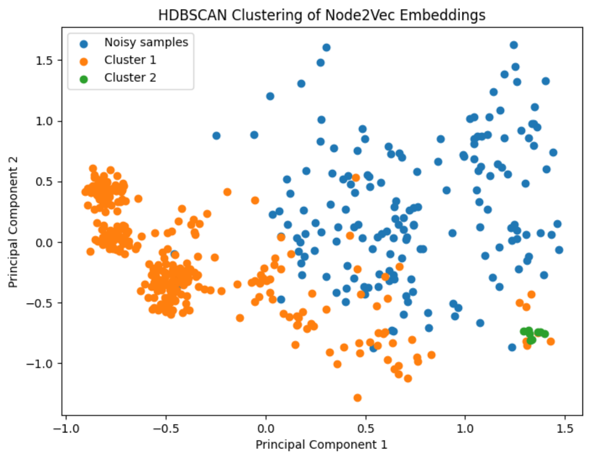
(a) A PCA transformation of the embeddings created by node2vec. The orange dots denote vectors of locations that are known to be MCAR. The blue dots are unknown.



(b) A PCA transformation of the embeddings created by node2vec, where each color denotes a cluster found by the k-means algorithm



(c) A PCA transformation of the embeddings created by node2vec, where each color denotes a cluster found by the dbscan algorithm

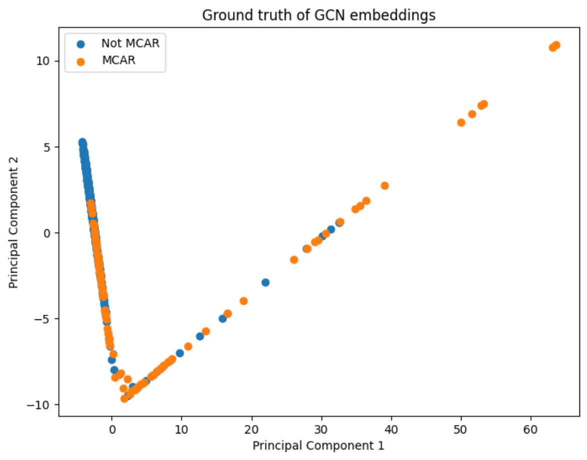


(d) A PCA transformation of the embeddings created by node2vec, where each color denotes a cluster found by the hdbscan algorithm

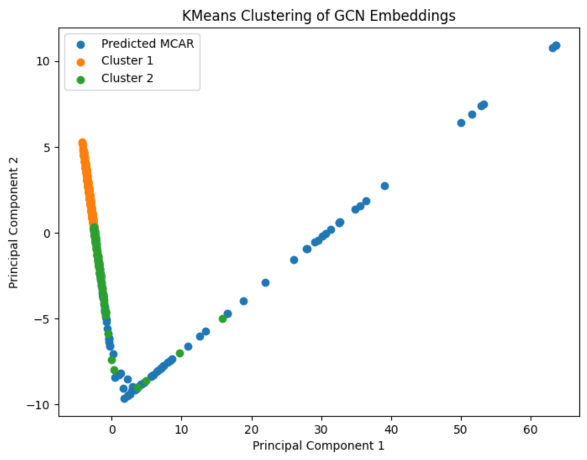
Fig. 7: PCA transformation of the embeddings created by node2vec. The plots show how the algorithms DBSCAN, HDBSCAN and K-means have clustered the embeddings.

data is known to be NMAR in a graph. Another implication worth considering is if this information could be used to detect if data is NMAR by examining the classification performance after imputation. We find that DBSCAN and HDBSCAN performed well at classifying MCAR as noise on the node2vec embeddings. This means that node2vec combined with either of the clustering algorithms could be used to classify MCAR. This is a novel finding that allows researchers to examine their datasets for MCAR as opposed to simply testing for MCAR. Given the efficiency and scalability of the model [31] and algorithms [30] selected in our method, our method will be suitable even for large graphs.

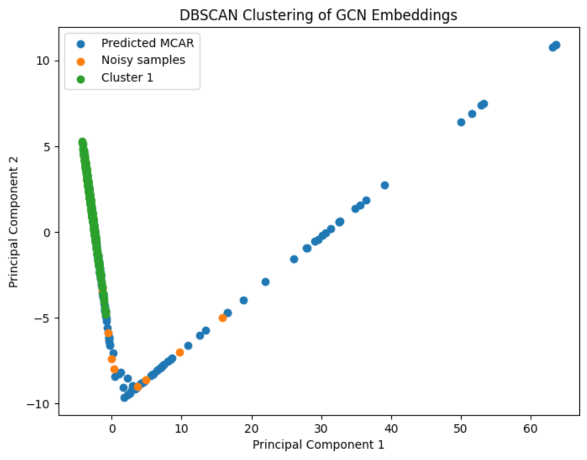
While the discoveries presented in this work are novel, we are not able to generalize our results, as we have only investigated one use case for each of our major contributions. As such, other and larger graphs and tasks would have to be employed to generalize our results. Another point that hinders generalizability is that we have not investigated or argued whether the simulations we employ of different mechanisms in graphs are equivalent to real-world missingness. It would require a study of how well the definitions we have used for missingness mechanisms fit real-world missingness. However, we found this to be outside the scope of what this work should encompass, and as such, it was not investigated. The link prediction models we selected do not improve the performance, which may be because they do not consider type information and consider the graph homogeneous. As such, valuable information is not leveraged. The models selected



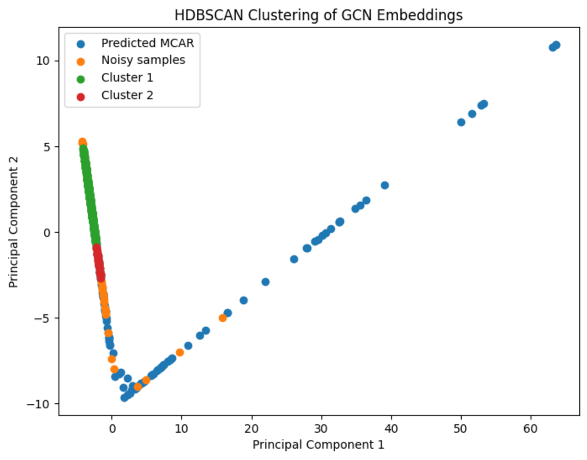
(a) A PCA transformation of the embeddings created by GCN. The orange dots denote vectors of locations that are known to be MCAR. The blue dots are unknown.



(b) A PCA transformation of the embeddings created by GCN, where each color denotes a cluster found by the K-means algorithm



(c) A PCA transformation of the embeddings created by GCN, where each color denotes a cluster found by the DBSCAN algorithm



(d) A PCA transformation of the embeddings created by node2vec, where each color denotes a cluster found by the HDBSCAN algorithm

Fig. 8: PCA transformations of the embeddings from GCN. Each figure displays a different clustering of locations depending on the clustering algorithm.

would need to be modified to incorporate the type information. Aside from the unaltered performance by using link prediction as imputation, we also found that models used on our MIMIC knowledge graph were very computationally expensive to train. For our task, training one model takes ≈30 hours on an NVIDIA A10 GPU.

While our results for MCAR classification using GCN show worse performance than node2vec with DBSCAN and HDBSCAN, the results are still promising. With an F5 score of 0.83, the result is substantial but not robust enough for use in practical applications. However, we can envision future work to improve these results. In the future, we could use a custom loss function, which encourages false positives. This could have resulted in a higher recall score, which is important for us. We also see positive silhouette scores when clustering on the negative embeddings created by the GCN model, however, we did not further evaluate the clusters. One way could be to inspect the clusters and determine if there is a common variable for the node in the clusters, such as a common type or relation. Another evaluation method would be to construct a graph and simulate MCAR, NMAR, and MAR for all nodes, such that labels are available for every node, but only using a subset of the MCAR labels during training for the GCN. As such, clusters could be evaluated with labels by using F1 score, recall, and precision. Currently, for DBSCAN and HDBSCAN, the embedding and clustering are two separate phases. As such, the training phase of the clustering algorithms cannot be used to update the learned embeddings. This complicates how we perform hyperparameter tuning. In our work, we only try one set of parameters for node2vec, compute the embeddings, and use grid search to determine the parameters for the clustering algorithms. One way to tune the parameters for node2vec using clustering could be to perform a grid search of the parameters for node2vec and use the mean silhouette score of the different clustering algorithms to determine the best parameters. However, this way the optimal parameters for clustering and embedding will depend on each other.

VIII. Conclusion

In this work, we propose a method for simulating missingness in a property graph and examine the impact of missingness mechanisms for the downstream task of prediction. We propose a method for imputing missing edges in a graph and three methods for determining the mechanism of missingness in a graph.

We examine how the missingness impacts the task of sepsis prediction, where we found the GraphSAGE model seems to be very robust towards missingness. Link prediction used as imputation did not show noteworthy improvements in F1 score when using Tucker and ConvE. However, for NMAR in particular, we found that the performance decreased for all missing rates, which was not the case with the other missing mechanisms. This could imply that imputation negatively affects the task when the mechanism is NMAR. For the clustering of the node2vec embeddings from the Wikidata graph, both DBSCAN and HDBSCAN show good results with an F5 score of 0.925 and 0.938 respectively for classifying MCAR as noise. When using a GCN for the binary classification of MCAR, the performance on the task results in an average F5 score of 0.826. When clustering the embeddings from the GCN the highest silhouette score is from DBSCAN with 0.76, which indicates a strong clustering where the clusters are distinct.

Our method for simulating different types of missingness allows practitioners to make a more exhaustive analysis of how their model performs in the presence of missingness. Our three methods for determining missingness mechanisms allow practitioners to gain more insight into their data, and discover patterns of missingness.

References

[1] A. Hansen, "Handling missing data for graph machine learning: A review," 2023.

[2] "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025," (Date last accessed 18-12-2023). [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created/

[3] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, pp. 127–163, 2000.

[4] S. Liu, S. Yao, Y. Huang, D. Liu, H. Shao, Y. Zhao, J. Li, T. Wang, R. Wang, C. Yang *et al.*, "Handling missing sensors in topology-aware iot applications with gated graph neural network," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 3, pp. 1–31, 2020.

[5] W. Zhong, Q. Suo, X. Jia, A. Zhang, and L. Su, "Heterogeneous spatio-temporal graph convolution network for traffic forecasting with missing values," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 707–717.

[6] A. Santos, A. R. Colaço, A. B. Nielsen, L. Niu, P. E. Geyer, F. Coscia, N. J. W. Albrechtsen, F. Mundt, L. J. Jensen, and M. Mann, "Clinical knowledge graph integrates proteomics data into clinical decision-making," *bioRxiv*, pp. 2020–05, 2020.

[7] S. Xiao, S. Wang, Y. Dai, and W. Guo, "Graph neural networks in node classification: survey and evaluation," *Machine Vision and Applications*, vol. 33, no. 1, p. 4, 2022.

[8] H. Zhang, Y. Hao, X. Cao, Y. Fang, W.-Y. Shin, and W. Wang, "Relation prediction via graph neural network in heterogeneous information networks with missing type information," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 2517–2526.

[9] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[10] H. Song and D. A. Szafir, "Where's my data? evaluating visualizations with missing data," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 914–924, 2018.

[11] Z. Chen, S. Tan, U. Chajewska, C. Rudin, and R. Caruna, "Missing values and imputation in healthcare data: Can interpretable machine learning help?" in *Conference on Health, Inference, and Learning*. PMLR, 2023, pp. 86–99.

[12] K. Mohan and J. Pearl, "Graphical models for processing missing data," *Journal of the American Statistical Association*, vol. 116, no. 534, pp. 1023–1037, 2021.

[13] D. C. Howell, "The treatment of missing data," *The Sage handbook of social science methodology*, pp. 208–224, 2007.

[14] Z. Chen, S. Tan, U. Chajewska, C. Rudin, and R. Caruna, "Missing values and imputation in healthcare data: Can interpretable machine learning help?" in *Conference on Health, Inference, and Learning*. PMLR, 2023, pp. 86–99.

[15] W.-C. Lin and C.-F. Tsai, "Missing value imputation: a review and analysis of the literature (2006–2017)," *Artificial Intelligence Review*, vol. 53, pp. 1487–1509, 2020.

[16] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 3840–3903, 2022.

[17] E. A. Bender and S. G. Williamson, *Lists, decisions and graphs*. S. Gill Williamson, 2010.

[18] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 international conference on management of data*, 2018, pp. 1433–1445.

[19] A. Hogan, *The Web of Data*. Springer, 2020.

[20] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[21] E. R. Hansen, T. Sagi, and K. Hose, "Diagnosis prediction over patient data using hierarchical medical taxonomies," 2023.

[22] A. Saeedi, E. Peukert, and E. Rahm, "Using link features for entity clustering in knowledge graphs," in *European Semantic Web Conference*. Springer, 2018, pp. 576–592.

[23] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE transactions on knowledge and data engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[24] M. Wang, L. Qiu, and X. Wang, "A survey on knowledge graph embeddings for link prediction," *Symmetry*, vol. 13, no. 3, p. 485, 2021.

[25] L. Tang and H. Liu, *Community detection and mining in social media*. Morgan & Claypool Publishers, 2010.

[26] "sklearn.cluster.kmeans¶," (Date last accessed 26-04-2024). [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[27] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[28] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2013, pp. 160–172.

[29] S. Chakraborty, N. K. Nagwani, and L. Dey, "Performance comparison of incremental k-means and incremental dbscan algorithms," *arXiv preprint arXiv:1406.4751*, 2014.

[30] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of data science*, vol. 2, pp. 165–193, 2015.

[31] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[32] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[33] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[34] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[35] D. Xu, H. Peng, C. Wei, X. Shang, and H. Li, "Traffic state data imputation: An efficient generating method based on the graph aggregator," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 13 084–13 093, 2021.

[36] X. Peng, J. Cheng, X. Tang, B. Zhang, and W. Tu, "Multi-view graph imputation network," *Information Fusion*, vol. 102, p. 102024, 2024.

[37] J. Li, X. S. Yan, D. Chaudhary, V. Avula, S. Mudiganti, H. Husby, S. Shahjouei, A. Afshar, W. F. Stewart, M. Yeasin *et al.*, "Imputation of missing values for electronic health record laboratory data," *NPJ digital medicine*, vol. 4, no. 1, p. 147, 2021.

[38] I. Spinelli, S. Scardapane, and A. Uncini, "Missing data imputation with adversarially-trained graph convolutional networks," *Neural Networks*, vol. 129, pp. 249–260, 2020.

[39] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. A. Celi, and R. Mark, "Mimic-iv," *PhysioNet. Available online at: https://physionet.org/content/mimiciv/1.0/(accessed August 23, 2021)*, pp. 49–55, 2020.

[40] "numpy.random.uniform," (Date last accessed 09-05-2024). [Online]. Available: https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html

[41] E. Getzen, L. Ungar, D. Mowery, X. Jiang, and Q. Long, "Mining for equitable health: Assessing the impact of missing data in electronic health records," *Journal of Biomedical Informatics*, vol. 139, p. 104269, 2023.

[42] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 1025–1035.

[43] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, M. Suo, P. Tillet, E. Wang, X. Wang, W. Wen, S. Zhang, X. Zhao, K. Zhou, R. Zou, A. Mathews, G. Chanan, P. Wu, and S. Chintala, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf

[44] I. Balažević, C. Allen, and T. M. Hospedales, “Tucker: Tensor factorization for knowledge graph completion,” *arXiv preprint arXiv:1901.09590*, 2019.

[45] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp, and J. Lehmann, “PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings,” *Journal of Machine Learning Research*, vol. 22, no. 82, pp. 1–6, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-825.html

[46] “Welcome to wikidata,” (Date last accessed 16-04-2024). [Online]. Available: https://www.wikidata.org/wiki/Wikidata:Main_Page

[47] “Statistics,” (Date last accessed 03-05-2024). [Online]. Available: https://www.wikidata.org/wiki/Special:Statistics

[48] “2014 sigkdd test of time award,” (Date last accessed 23-05-2024). [Online]. Available: https://www.kdd.org/News/view/2014-sigkdd-test-of-time-award

[49] “node2vec,” (Date last accessed 22-05-2024). [Online]. Available: https://github.com/eliorc/node2vec

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[51] P. J. Rousseeuw, “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis,” *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

Acronyms

BFS	breadth first search. 4
DFS	depth first search. 4
DGL	deep graph learning. 9
EBM	Explainable Boosting Machines. 1, 2
EM	expectation maximization. 2
GAT	graph attention network. 1
GCN	graph convolutional network. 1, 2, 4, 7–10, 12, 13
GMM	gaussian mixture model. 2
GNN	graph neural network. i, 1, 2, 4
KNN	K-nearest neighbors. 2
MAR	missing at random. 2, 5–10, 12
MCAR	missing completely at random. 1, 2, 5–13
MICE	multiple imputation by chained equations. 2
MIMIC	Medical Information Mart for Intensive Care. 12
MIMIC-IV	Medical Information Mart for Intensive Care IV. 6, 7
MLP	multilayer perceptron. 2
NMAR	not missing at random. 2, 5–13
PCA	principal component analysis. 10–12
RF	random forest. 2