Evaluation of whole graph embedding techniques for a clustering task in the manufacturing domain

Bewertung von Techniken zur Grapheneinbettung für eine Clustering-Aufgabe im Produktionsbereich

Master-Thesis

Autor: Yusef Iskandar
Matrikelnummer: 2914796
Betreuer: Prof. Dr.-Ing. Joachim Metternich | Beatriz Bretones Cassoli, M. Sc.
Abgabe: Darmstadt, 05.09.22

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Master Thesis
für
Yusef Iskandar | 2914796

Institut für Produktionsmanagement, Technologie und Werkzeugmaschinen
Fachbereich Maschinenbau
Otto-Berndt-Str. 2
64287 Darmstadt

Telefon: 06151 16-20080
Telefax: 06151 16-20087

Thema:
Bewertung von Techniken zur Grapheinbettung für eine Clustering-Aufgabe im Produktionsbereich

Topic:
Evaluation of whole graph embedding techniques for a clustering task in the manufacturing domain

Hintergrund:
Kontextinformationen ermöglichen Anwendungen der künstlichen Intelligenz (KI) den Umgang mit Ambiguität. So können Vorhersagen verbessert und Entscheidungsprozesse unterstützt werden. Semantische Technologien wie Wissensgraphen (*Knowledge Graphs*) erfassen Kontext- und Domänenwissen, und können damit zur Erklärbarkeit und Akzeptanz von KI bei der Realisierung flexibler Fertigungssysteme beitragen. Grapheinbettungen (*Graph Embeddings*), bei denen es sich um Vektordarstellungen von Wissensgraphen handelt, ermöglichen die Verwendung von Wissensgraphen als Input für Algorithmen des maschinellen Lernens und die Durchführung von Datenanalyseaufgaben wie Clustering, Klassifizierung und Regression. Die Auswahl einer geeigneten Technik ist dabei entscheidend für die Erzeugung aussagekräftiger Einbettungen für eine bestimmte Aufgabe.

Aufgabenstellung:
Ziel der Arbeit ist es, den Einsatz von Grapheinbettungen in einem Anwendungsfall in der Produktion zu evaluieren. Die für die Evaluierung verwendeten Datensätze sind bei Kaggle öffentlich verfügbar und stammen von Produktionslinien der Firma Bosch. Die Aufgabe besteht darin, Grapheinbettungen für jedes an der Produktionslinie montierte Produkt zu generieren und diese zu verwenden, um Produkte derselben Variante zu clustern. Der Hauptbeitrag dieser Arbeit ist die Bewertung und der Vergleich der neuesten Techniken zur Grapheinbettung - namentlich Graph2Vec, NetLSD, WaveletCharacteristic, IGE, LDP, GeoScattering, GL2Vec, SF und FGSD.

Im Rahmen dieser Arbeit werden die folgenden Punkte behandelt:

- Literaturrecherche des Stands der Technik zur Einbettung ganzer Graphen und Identifizierung geeigneter Vergleichsmetriken
- Explorative Datenanalyse des Bosch-Produktionsdatensatzes und Definition des Graphdatenmodells
- Konzeption einer Datenverarbeitungspipeline zur Generierung der Grapheinbettungen aus den tabellarischen Rohdaten
- Aufbau, Training und Validierung von Grapheinbettungen mit den neuesten Techniken zur Einbettung ganzer Graphen
- Quantitativer Vergleich der Leistung der Techniken für die Clustering-Aufgabe
- Dokumentation und Präsentation der Ergebnisse

Beginn: 01.04.2022
Ende: 01.09.2022
Betreuer: Beatriz Bretones Cassoli, M.Sc.

Prof. Dr.-Ing. J. Metternich

Erklärung zur Abschlussarbeit gemäß §22 APB und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Yusef Iskandar, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.
Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

(Ort, Datum)	(Unterschrift)

Folgende Einverständniserklärung ist unabhängig vom Prüfungsverfahren zur Masterprüfung (ein Exemplar verbleibt bei den Prüfungsakten) und ohne Einfluss auf die Bewertung der Master-Thesis.
Ich bin damit einverstanden, dass die Master-Thesis in den Bibliotheksbestand der TU Darmstadt aufgenommen wird und öffentlich zugänglich gemacht wird.

(Ort, Datum)	(Unterschrift)

Die TU Darmstadt bittet Sie im Interesse eines freien Informationsaustausches, ihr Urheberrecht an der Arbeit zu wissenschaftlichen Zwecken nutzen zu dürfen. Sie können die Nutzung Ihres Urheberrechts durch die TU Darmstadt ohne Angabe von Gründen und ohne nachteilige Folgen für die Bewertung der Arbeit verweigern.

Ich bin damit einverstanden, dass die TU Darmstadt das Urheberrecht an meiner Master-Thesis zu wissenschaftlichen Zwecken nutzen kann.

(Ort, Datum)	(Unterschrift)

Abstract

Key Words: Graph Mining, Knowledge Graph, Feature Engineering

Production systems in manufacturing consume and generate data. Representing the relationships between subsystems and their associated data is complex, but suitable for Knowledge Graphs (KG), which allow us to visualize the relationships between subsystems and store their measurement data. In this work, KG act as a feature engineering technique for a clustering task by converting KG into Euclidean space with so-called graph embeddings and serving as input to a clustering algorithm. The Python library Karate Club proposes 10 different technologies for embedding whole graphs, i.e., only one vector is generated for each graph. These were successfully tested on benchmark datasets that include social media platforms and chemical or biochemical structures. This work presents the potential of graph embeddings for the manufacturing domain for a clustering task by modifying and evaluating Karate Club's techniques for a manufacturing dataset. First, an introduction to graph theory is given and the state of the art in whole graph embedding techniques is explained. Second, the Bosch production line dataset is examined with an Exploratory Data Analysis (EDA), and a graph data model for directed and undirected graphs is defined based on the results. Third, a data processing pipeline is developed to generate graph embeddings from the raw data. Finally, the graph embeddings are used as input to a clustering algorithm, and a quantitative comparison of the performance of the techniques is conducted.

This work is licensed under a Creative Commons Attribution 4.0 International License
https://creativecommons.org/licenses/by/4.0/

Table of Contents

ERKLÄRUNG ZUR ABSCHLUSSARBEIT GEMÄß §22 APB UND § 23 ABS. 7 APB DER TU DARMSTADT 1

ABSTRACT 2

TABLE OF CONTENTS 3

LIST OF FIGURES 6

LIST OF TABLES 8

LIST OF SYMBOLS 9

1. INTRODUCTION 11

1.1 Motivation 11

1.2 Research Question 11

1.3 Methodology...11

1.4 Thesis Outline 12

2 STATE OF THE ART 14

2.1 Graph Theory 14

2.1.1 Graphs...14

2.1.2 Sparse Matrices...18

2.1.3 Laplacian Matrix...21

2.2 Graph Embeddings 22

2.2.1 Graph2Vec...24

2.2.2 Graph and Line Graph to Vector (GL2vec) 27

2.2.3 Family of Graph Spectral Distances (FGSD)...29

2.2.4 Network Laplacian Spectral Descriptor (NetLSD)..31

2.2.5 Spectral Features (SF)..34

2.2.6 Invariant Graph Embedding (IGE) 36

2.2.7 Local Degree Profile (LDP)...38

2.2.8 Geo Scattering...39

2.2.9 Feather-Graph 42
2.2.10 Wavelet Characteristic 45
2.3 Clustering 48
2.3.1 K-means 48
2.3.2 Performance Evaluation 50
3 DATASET 54
3.1 Overview Bosch Dataset 54
3.1.1 Application Domain 54
3.1.2 Dataset Overview 54
3.1.3 Bosch's Objective 56
3.2 Exploratory Data Analysis 56
3.2.1 Numerical Features 57
3.2.2 Date Features 61
4 EXPERIMENT 65
4.1 Preparation 65
4.1.1 Technical Environment 65
4.1.2 Data Preprocessing 66
4.2 Workflow 68
4.3 Graph Model 69
4.3.1 Data Model 69
4.3.2 Directed Graph Model 70
4.3.3 Undirected Graph Model 70
4.3.4 Implementation 71
4.4 Implementation of Graph Embeddings 72
4.4.1 Graph Orientation 73
4.4.2 Node Attributes 74
4.5 Clustering Task 74
5 EVALUATION AND DISCUSSION 76

5.1 Comparison of Graph Embedding Techniques **76**

5.2 Results and Discussion **77**

5.2.1 Baseline Algorithm 77

5.2.2 Results and Discussion 78

5.3 Summary **81**

6 CONCLUSION AND OUTLOOK **82**

6.1 Conclusion **82**

6.2 Outlook **83**

BIBLIOGRAPHY .... **I**

List of Figures

Figure 1. CRISP-ML(Q) diagram inspired by [6] 12

Figure 2. Diagram of graph G 14

Figure 3. Different types of directed graphs....16

Figure 4. Bipartite graph....17

Figure 5. Degree matrices for a directed and undirected graph....17

Figure 6. Directed graph and corresponding adjacency matrix....18

Figure 7. Conversion of a dense matrix to a sparse matrix in COO format [16] 19

Figure 8. Conversion of a dense matrix to a sparse matrix in CSR format [16] 20

Figure 9. Conversion of a dense matrix to a sparse matrix in CSC format [16] 20

Figure 10. Example of left and right normalized Laplacian matrices [17] 22

Figure 11. Example of left and right normalized Laplacian matrices for directed graphs [17] 22

Figure 12. Categorization of Karate Club's whole graph embedding techniques....24

Figure 13. WL relabeling for graph G [25] 26

Figure 14. Conversion of graph G to line graph LG [25]....28

Figure 15. Graph generation model of FGSD [29]....30

Figure 16. Schematic illustration of the SF model. $\mathcal{L}$ denotes the normalized Laplacian and c the predicted class. [30]....35

Figure 17. Example of Daubechies wavelets with N = 2, 4, 8 [35] 39

Figure 18. Operations of the wavelet scattering transform inspired by [37]....40

Figure 19. Architecture for using Geo Scattering of graph G and signal x [36] 41

Figure 20. r-scale random walk weighted characteristic functions for a low and high degree node [38]....45

Figure 21. Example of clustering procedure with k-means algorithm [42]....49

Figure 22. Nomenclature for numerical features 55

Figure 23. Number of products classified as functioning or failed 57

Figure 24. Number of measurements for each production line....58

Figure 25. Number of products passed through each station 59
Figure 26. Number of product failures for each station 59
Figure 27. Error Rate for each station 60
Figure 28. The proportion of NaN values vs. Feature 61
Figure 29. Number of records for date features over time 62
Figure 30. Sensor Time Autocorrelation 62
Figure 31. Violin Plot on total time spent on a product for different product classes 64
Figure 33. File Structure of the software implementation 66
Figure 34. Workflow Data Preprocessing 67
Figure 35. Flowchart of the workflow after data preprocessing 69
Figure 36. Data model for the Knowledge Graph [5] 69
Figure 37. Example of the directed graph model 70
Figure 38. Excerpt from the first 15 nodes of product id 4's graph 72

List of Tables

Table 1. Assignment of stations to their corresponding line 55

Table 2. Total time spent on a product for different product classes 63

Table 3. Excerpt from numerical training set..67

Table 4. Excerpt from the numerical CSV file of product Id 4...67

Table 5. Comparison of graph embedding techniques by selected metrics 77

Table 6. Clustering results for 10 graph embedding techniques vs. baseline for four different evaluation metrics..78

Table 7. Performance for GeoScattering, Feather-Graph and Wavelet Characteristic using different feature vectors 80

List of Symbols

Symbol	**Meaning**
A	adjacency matrix
D	degree matrix
E	set of edges
G	graph
I	identity matrix
L	Laplacian matrix
λ	embedding size
$L(G)$	line graph of G
P	transition matrix
V	set of vertices
W	weight matrix

Abbreviations

AI	Artificial Intelligence
AMI	Adjusted Mutual Information
ARI	Adjusted Rand Index
COO	Coordinate Form
CRISP	Cross-Industry Standard Process
CSC	Compressed Sparse Column
CSR	Compressed Sparse Row
EDA	Exploratory Data Analysis
FGSD	Family of graph spectral distances
GL2Vec	Graph and line graph to vector
IDE	Integrated Development Environment
IGE	Invariant graph embedding
KG	Knowledge Graph

LDP	Local degree profile
MDPA	Minimum difference of pair assignments
MI	Mutual Information
MiP	Management of Industrial Production
ML	Machine Learning
NetLSD	Network Laplacian spectral descriptor
PCA	Principal Component Analysis
PTW	Institute of Production Management, Technology and Machine Tools
RI	Rand Index
SF	Spectral features
SC	Silhouette Coefficient
TUDa	Technical University of Darmstadt
WL	Weisfeiler-Lehman

1. Introduction

At the beginning of this thesis, the motivation of the research topic is outlined. Subsequently, the research question is formulated and the methodology as well as the structure of the work are presented.

1.1 Motivation

Industry 4.0 is revolutionizing the manufacturing domain. Especially for the purpose of Industry 4.0, Artificial Intelligence (AI) powered manufacturing is a necessity. According to a study by Google Cloud in 2021, 64% of manufacturers rely on AI to support day-to-day operations, and the area where it is currently deployed the most is quality control [1]. The reason for the massive focus on AI projects is that manufacturing companies can achieve up to 30% yield improvements and 15% waste reduction, based on IBM Research [2].

Through contextual information, AI applications can deal with ambiguity to enhance predictions and support decision-making processes [3]. Knowledge Graphs (KG) capture context and domain knowledge and can thus contribute to the explainability and acceptance of AI in the realization of flexible manufacturing systems [4]. Graph embeddings, which are vector representations of KG, allow KG to be used as inputs for Machine Learning (ML) algorithms and the realization of data analysis tasks such as clustering and classification [5]. In this context, the selection of an appropriate graph embedding technique is crucial for generating meaningful embeddings for a given task.

1.2 Research Question

The research question of this work is formulated as: "How effective is the use of graph embeddings for performing a clustering task in the manufacturing domain?". The goal is to understand how graph embeddings perform as a feature engineering technique for downstream ML tasks such as clustering. It also aims to test how well the graph data format is suited for representing production data and whether it has advantages compared to other techniques.

1.3 Methodology

Many ML and data science projects are not well structured, and their results are not reproducible. Therefore, the Cross-Industry Standard Process for the development of Machine Learning applications with Quality assurance (CRISP-ML (Q)) methodology was proposed. This thesis uses the CRISP-ML methodology in Figure 1 as a guideline for solving the research question. [6]

The starting phase "Business and Data Understanding" is about defining business goals and translating them into ML objectives, collecting and verifying data quality and finally assessing the feasibility of the project. The second phase aims at creating a dataset for the subsequent modeling

phase. The modeling phase itself is the ML-specific phase that seeks to specify one or more ML models. Afterwards, the ML workflow is integrated into a pipeline to enable repeatable model training. In the evaluation step, the performance of the trained model is validated using a test set. Based on success criteria of the evaluation step, the decision on deployment is made. The deployment phase of a ML model is characterized by its integration into the existing software system. After completion of the evaluation step, the ML model is evaluated for the use in the production environment. Once the ML model is deployed in the production process, it is important to monitor its performance and maintain it. However, the deployment and monitoring & maintenance steps are not covered in this work. [6]

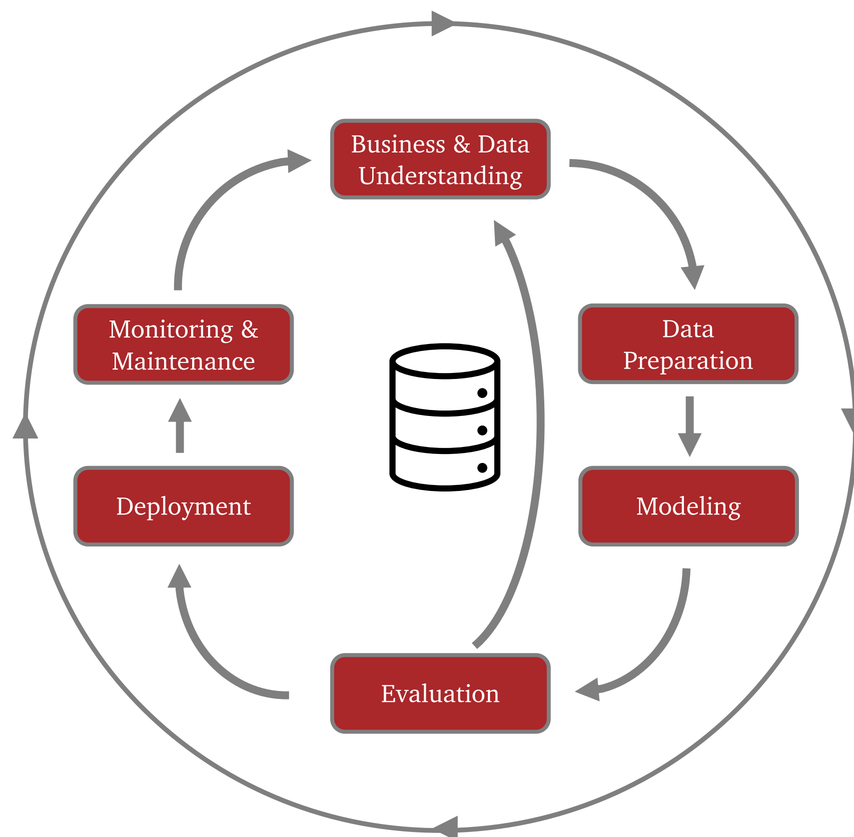


Figure 1. CRISP-ML(Q) diagram inspired by [6]

1.4 Thesis Outline

This thesis starts with an introduction to graph theory, providing the mathematical background necessary to understand the structure of graph embedding techniques. Chapter 2 analyzes the state of the art for whole graph embeddings to transform graph structures into vector space while maximally preserving their properties. For this purpose, ten methods for embedding whole graphs from the Karate Club library are used. Additionally, the k-means clustering algorithm and clustering performance metrics are presented.

Chapter 3 provides an Exploratory Data Analysis (EDA) performed on a Bosch production line dataset, which forms the use case for evaluating graph embedding techniques. In Chapter 4, the experiment was conducted to design a data processing pipeline to generate graph embeddings from the raw data. This included preprocessing the data, defining and creating graphs, and customizing the implementation of the graph embedding techniques. Finally, the clustering task was performed, and in Chapter 5, a quantitative comparison of the performance of the techniques was made, and the results were discussed. Finally, a conclusion was drawn and an outlook on future tasks is given in Chapter 6.

2 State of the Art

This work is based on a fundamental understanding of graph theory, graph embeddings, and unsupervised learning methods. The State of the Art chapter introduces the necessary graph embedding techniques investigated in this thesis and provides an overview of suitable comparison metrics. It forms the basis for understanding the scientific background for the proposed Experiment described in the following chapters.

2.1 Graph Theory

This section covers fundamental definitions and properties of graphs, introduces the notion of sparse matrices for efficient graph representation, and defines the Laplacian matrix, which plays an important role for the graph embedding techniques discussed in this work.

2.1.1 Graphs

A graph G is defined as an ordered pair $G = (V, E)$, where V is a set of vertices (or nodes) and $E \subseteq (V \times V)$ be a set of edges (or lines). These represent nodes as entities, and the relationships between nodes are expressed by edges. Graphs are usually portrayed by drawing a point for each vertex and joining two points by a line when the two vertices form an edge. The node set of the graph G is denoted as $V(G)$ and its edge set as $E(G)$. For clarity, Figure 2 shows a graph G with its corresponding vertex set $V(G) = \{v_1, v_2, v_3, v_4\}$ and edge set $E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6\}$. The following explanations and definitions assume that the mentioned graphs are non-empty. [7,8]

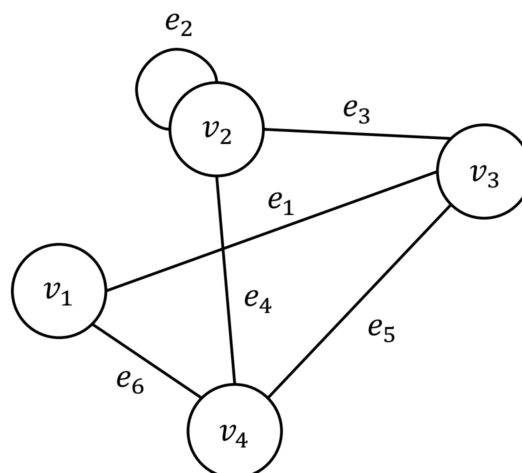


Figure 2. Diagram of graph G

Directed Graph. When the edges of a graph have an orientation, they are called directed graphs, otherwise, they are called undirected graphs. A directed graph (also called digraph) is defined as

TU DARMSTADT

a pair (V, E) of vertex and edge sets, where two functions $init: E \to V$ and $ter: E \to V$ assign to every edge e an initial vertex $init(e)$ and a terminal vertex $ter(e)$. The edge e is called directed from $init(e)$ to $ter(e)$. According to the above definition a directed graph can have several edges between two vertices v_1 and v_2. These edges are called *multiple edges*, and if they lead from the same node to the other, the edge e is called a loop. [7]

Incident and adjacent. Two vertices connected by an edge are called its ends. An edge e is incident to its ends, that is, if $v \in e$. If two vertices v_1, v_2 of G are adjacent (or neighbors), then $\{v_1, v_2\}$ is an edge of G. Two edges can also be adjacent if they have a common end. A graph is called *complete*, if every pair of distinct vertices is connected by a unique edge. An example of a complete graph with $n = 3$ vertices is a triangle. [7]

Graph isomorphism. Two graphs G_1 and G_2 are declared identical if they can be represented by an identical diagram. However, it is possible for two non-identical graphs to have essentially the same diagram, except that they have different node and edge labels. Then the two graphs G_1 and G_2 are not identical, but isomorphic. A graph $G_1 = (V_1, E_1)$ is isomorphic to $G_2 = (V_2, E_2)$, also denoted as $G_1 \simeq G_2$, if there exists a bijective[1] function $\varphi: V_1 \to V_2$ such that $(\varphi(u), \varphi(v)) \in E_2$ for each $(u, v) \in E_1$, which simply means that the nodes of graph G_1 are mapped to nodes of graph G_2 by a bijective function φ. [7,8]

Subgraph. G' is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$, denoted as $G' \subseteq G$. In other words, G contains G' (or G is a supergraph of G').[7]

Path. A path is a finite or infinite sequence of edges connecting a sequence of distinct vertices. A path can be denoted as a graph $P = (V, E)$ of the form

$$V = \{x_0, x_1, \dots, x_k\}, \qquad E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}$$

where the vertices x_0 and x_k are connected by the path P. The parameter k describes the number of edges of the path, i.e., the length of the path and is denoted by P^k. A path is often referred to as a natural sequence of its vertices, where $P = x_0x_1 \dots x_k$ is a path between x_0 and x_k. [7]

Walk. A walk of length k is an alternating sequence $v_0v_1 \dots v_k$ of vertices and edges in G with the edges $e_i = \{v_i, v_{i+1}\}$ for all $i < k$. The difference to a path is that the vertices and edges in a walk are not all distinct. However, it can be said that every walk between two vertices contains a path between these vertices. [7]

A *random walk* is a special case, where all steps $v_0v_1 \dots$ are chosen randomly and allow $v_i = v_{i+1}$. The probability of a random walk that started in node i to be in node j after k steps can be represented by the transition matrix P_{ij}^k. With the assumption that if G is a digraph, then $\deg^+(v_j) > 0$ for every vertex v, the transition matrix P_{ij} can defined as: [9,10]

[1] is a function between the elements of two sets, where each element of one set is paired with exactly one element of the other set, and vice versa [57]

$$P_{ij} := \begin{cases} \frac{1}{\deg(v_j)} & \text{if } (v_i, v_j) \text{ is an edge in the graph } G \\ \frac{1}{\deg^{+}(v_j)} & \text{if } (v_i, v_j) \text{ is an edge in the digraph } G \\ 0 & \text{otherwise.} \end{cases}$$

Connectivity. In an undirected graph G, two vertices v_1 and v_2 are called connected if G contains a path from v_1 to v_2. In general, the graph G is connected if two of its vertices are linked by a path in G. Otherwise, they are called disconnected. A special case is a graph with only one vertex that is connected, while an edgeless graph with more than one vertex is disconnected. Directed graphs are distinguished between weakly connected, unilaterally connected and strongly connected graphs. A directed graph is weakly connected if replacing all its directed edges with undirected edges produces a connected graph. The graph is unilaterally connected if it contains a directed path from v_1 to v_2 or a directed path from v_2 to v_1 for each pair of vertices v_1 and v_2. Finally, a directed graph is defined as strongly connected if it contains a directed path from v_1 to v_2 and a directed path from v_2 to v_1 for every pair of vertices v_1 and v_2. The different graph connection types are depicted in Figure 3. [7,11]

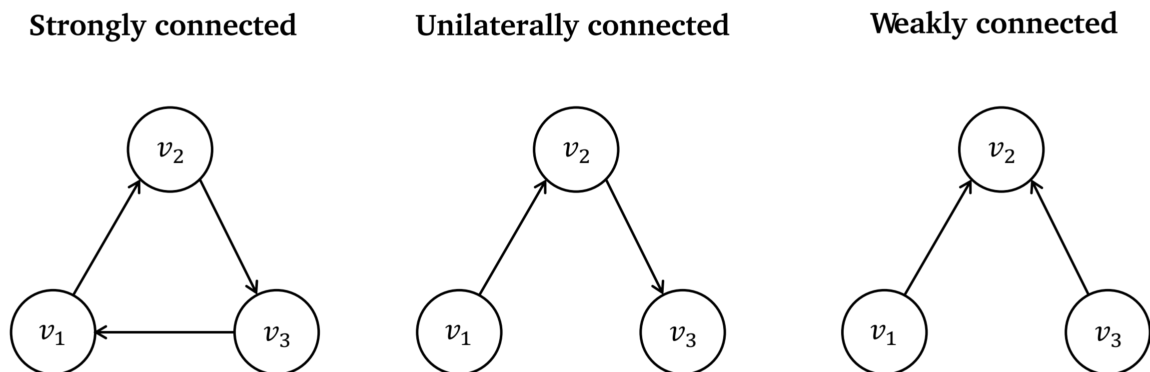


Figure 3. Different types of directed graphs

Bipartite graph. A graph $G = (U, V, E)$ is called bipartite if its vertices can be divided into two disjoint and independent sets U and V, where each edge connects a vertex in U to one in V (see Figure 4). [7,12]

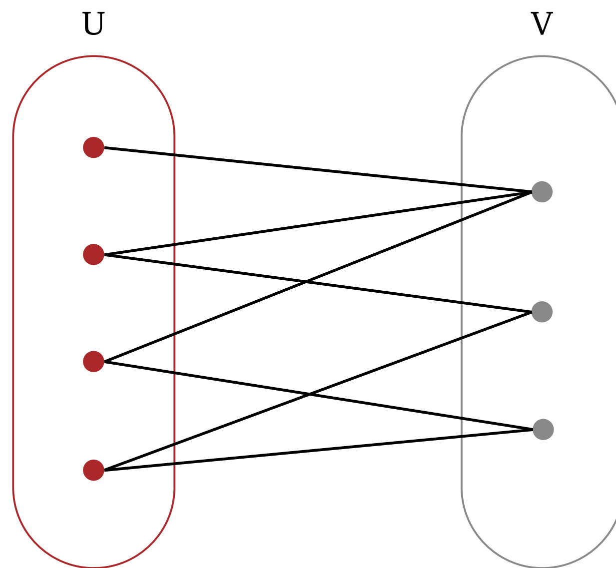


Figure 4. Bipartite graph

Degree Matrix. The degree $d_G(v) = d(v)$ of a vertex v is the number $|E(v)|$ of edges on v, which is equal to the number of neighbors of v. By knowing this, we define the diagonal matrix $(d_{ij})_{n\times n}$ with $d_{ii} = d(v_i)$ and $d_{ij} = 0$ otherwise. The definition refers to undirected graphs. However, for directed graphs, there is an in-degree and out-degree matrix that count the number of edges at vertex v that go either in or out (see Figure 5). The figure also shows the degree matrix for the case that the graph is treated as undirected. [7,13]

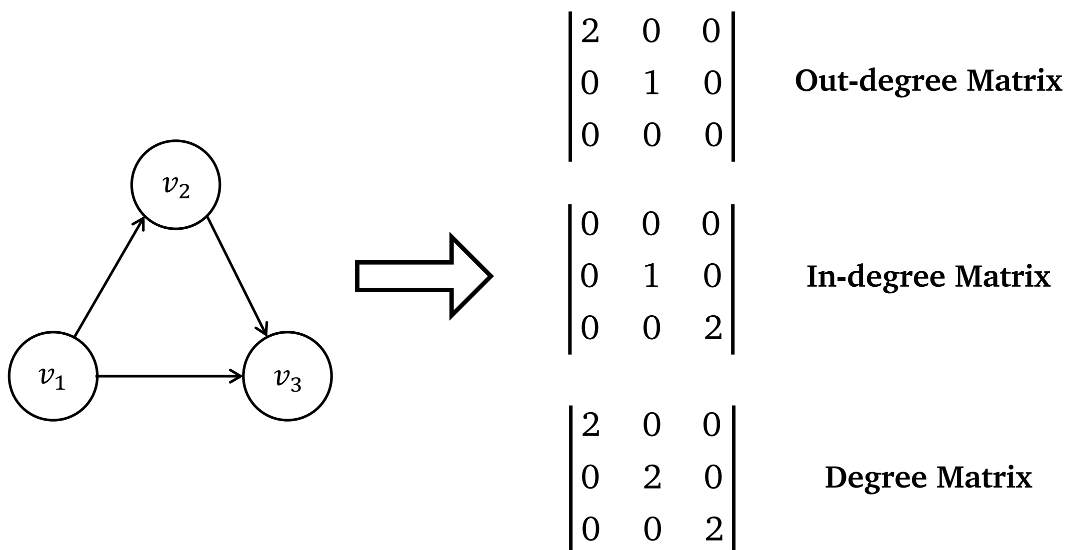
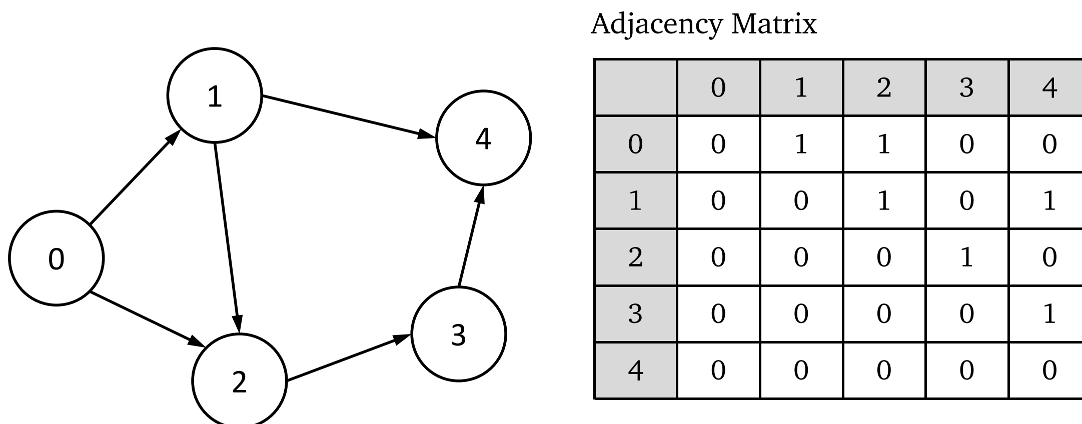


Figure 5. Degree matrices for a directed and undirected graph

Adjacency Matrix. The adjacency matrix $A = (a_{ij})_{n \times n}$ is a square matrix used to describe a graph G. The elements in the matrix indicate whether pairs of vertices in the graph are adjacent or not. If $v_i v_j \in E$, then $a_{ij} = 1$, otherwise $a_{ij} = 0$. The adjacency matrix of an undirected graph is symmetric. Figure 6 shows a directed graph with its corresponding asymmetric adjacency matrix. [7]



Adjacency Matrix

	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

Figure 6. Directed graph and corresponding adjacency matrix

A graph whose edges are assigned a numerical weight is known as a weighted graph. The number assigned to each edge e is the weight of e, denoted by $w(e)$. The weights are represented by a weight matrix $W = (w_{ij})_{n \times n}$. The weight matrix of an unweighted graph is equal to its adjacency matrix. [14]

By examining the adjacency or weight matrix, we see that that these matrices can be very large and contain many values that are equal zero. To save computation time and memory, other data structures have been developed that store the same values of these matrices in a different way. Therefore, the concept of sparse matrices is explained below.

2.1.2 Sparse Matrices

A sparse matrix is a matrix in which most of the elements are equal to zero. A matrix can be called sparse if more than 50% of its values are zero. Otherwise, if the number of non-zero entries dominates, then these matrices are called dense. Sparse matrices are able to perform faster operations and use less memory than dense matrices. These properties are very beneficial when working with large datasets in data science. The implementation of the sparse matrices is provided by the Python package scipy sparse. [15,16]

2.1.2.1 Coordinate Format

The simplest sparse format is the COOrdinate (COO) format (see Figure 7). It uses three different subarrays to store the values and their corresponding positions. The first two arrays contain location information about the row and column number of the value to be stored. The third array stores the actual value. For some datasets with an increasing number of data points, the COO format is a great choice because each data point is stored with three entries, while for instance an additional data point in a dense 6×6 matrix stores a value with six entries. [16]

One advantage of this sparse format is that it facilitates fast conversion to and from other sparse formats like Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC) which are explained below. [15]

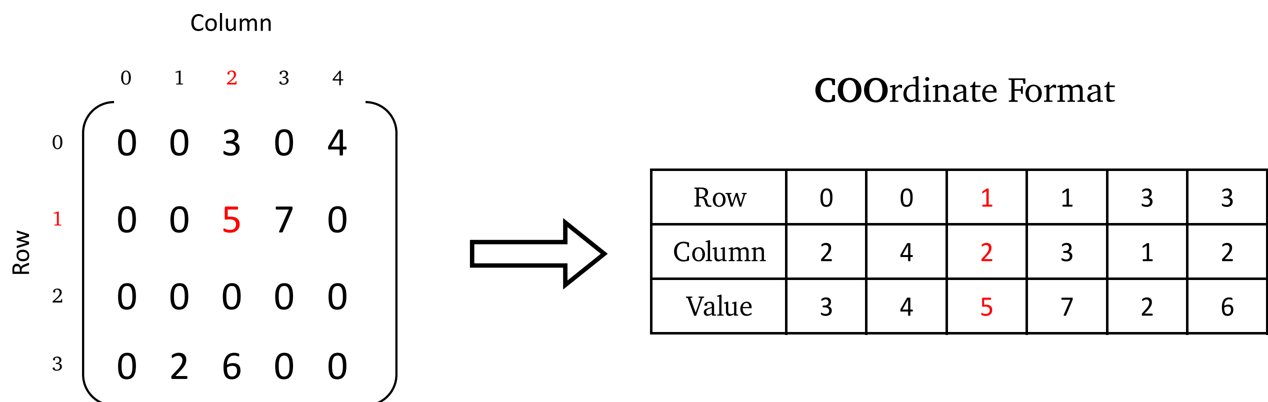


Figure 7. Conversion of a dense matrix to a sparse matrix in COO format [16]

2.1.2.2 Compressed Sparse Row Format

The CSR or Yale format represents a matrix similar to the COO format, with three different subarrays, that respectively contain the compressed row indices, the column indices, and the non-zero values. While the COO format represents the row indices, CSR compresses them, hence the name. Figure 8 shows how a dense matrix was converted into a sparse matrix in CSR format. The algorithm iterates over each row of the dense matrix. Note that the first value of the row pointer is always 0, and the last is always the number of non-zero values. The first non-zero value identified is 8 at column index 0 which can be checked in the figure’s table. The second non-zero value is 2 with column index 2. The row pointer can be considered as a numeric counter that adds a +1 for each non-zero value detected. Since the value 2 is the second detected non-zero value, the corresponding row pointer value is 2. As highlighted in Figure 8, the row pointer value for 5 is 3 because it is the third non-zero value. The algorithm continues this procedure by inspecting row by row. The advantage of the format is that it allows fast row access and matrix multiplication. [12,16]

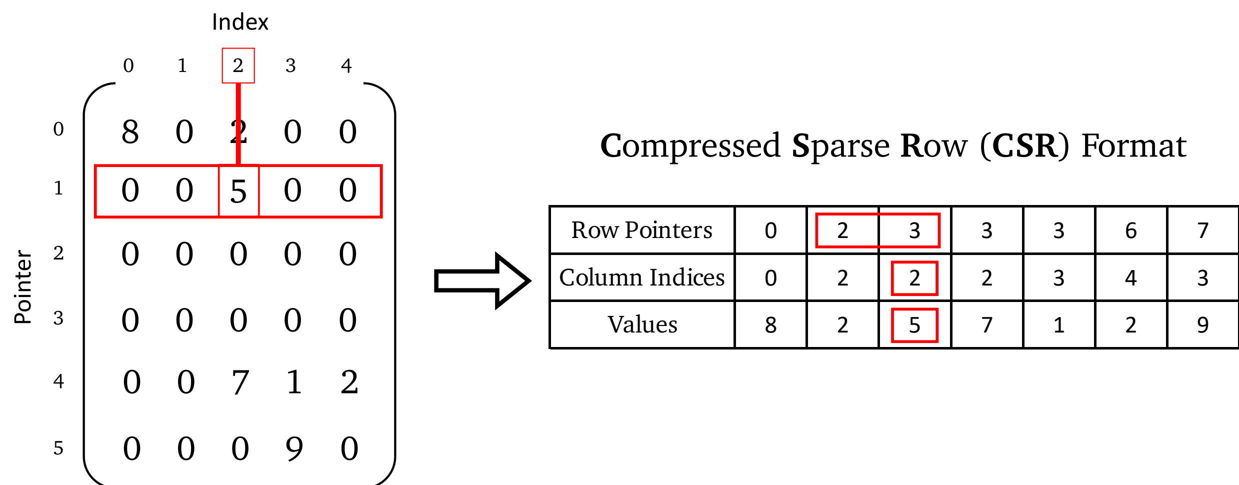


Figure 8. Conversion of a dense matrix to a sparse matrix in CSR format [16]

2.1.2.3 Compressed Sparse Column Format

The algorithm of CSC works similarly to CSR, but instead has column-based pointers and row indices (see Figure 9). The algorithm iterates through all columns. As shown in the figure, the values 2, 5, 7, which represent the second, third and fourth non-zero values of the matrix, are all in one column. As the pointer is updated through all entries in a column after iteration, the column pointer for 2, 5, and 7 jumps from 1 to 4 as three non-zero values are added. Note in this example that the CSC format is slightly more compact with one less index pointer than CSR. While CSR allows fast row access, CSC makes column access very efficient. [16]

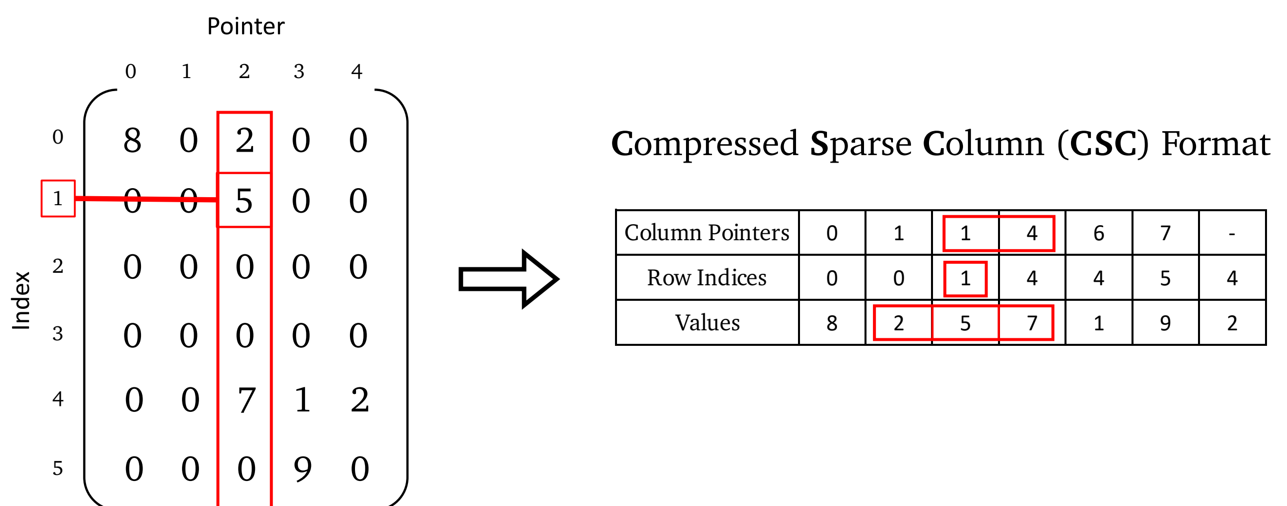


Figure 9. Conversion of a dense matrix to a sparse matrix in CSC format [16]

2.1.3 Laplacian Matrix

The Laplacian matrix is a matrix representation of a graph, denoted by L. For an undirected graph $G = (V, E)$ with n vertices, the Laplacian matrix $L_{n\times n}$ is defined as

$$L_{i,j} := \begin{cases} d_i & \text{if } v_i = v_j \\ -1 & \text{if } v_i v_j \in E \\ 0 & \text{otherwise.} \end{cases}$$

The equivalent is $L = D - A$, where D is the degree matrix and A the adjacency matrix of the graph. [17]

For directed graphs we must consider that there are two different degree matrices, which is why directed graphs have an out-degree Laplacian $L_{out} = D_{out} - A$, and an in-degree Laplacian matrix $L_{in} = D_{in} - A$. Note that the adjacency matrix of a directed graph is not necessarily symmetric, which means that the corresponding Laplacian matrix can also be asymmetric. The Laplacian matrix is widely used in spectral graph theory. This theory relates the properties of a graph to a *spectrum* that consists of the eigenvalues of matrices associated with the graph, such as its Laplacian matrix. [17]

Normalization. The goal of normalization is to make the diagonal entries of the Laplacian matrix all unit. There are two ways to normalize an existing Laplacian matrix.

The first way generates the symmetrically normalized Laplacian which is defined as

$$L^{sym} := (D^+)^{1/2} L\, (D^+)^{1/2} = I - (D^+)^{1/2} A (D^+)^{1/2}, \tag{2.1}$$

where D^+ is the Moore-Penrose inverse, which is referred to the pseudoinverse of D. The symmetrically normalized Laplacian matrix is symmetric if the adjacency matrix is symmetric. For a non-symmetric adjacency matrix of a directed graph, there can be again be two matrices, either the Laplacian matrix normalized to the outer degree or the Laplacian matrix normalized to the inner degree. [17]

The second way to normalize the Laplacian matrix is to create left (random walk) and right normalized Laplacians. The left (random walk) normalized Laplacian matrix is defined as

$$L^{rw} := D^+ L = I - D^+ A, \tag{2.2}$$

where D^+ describes again the Moore-Penrose inverse. The right normalized Laplacian is calculated in the same way, except that AD^+ is used instead of D^+A. When the adjacency matrix is symmetric, the left and right normalized Laplacians are generally not symmetric. Figure 10 shows an example of a random walk and right normalized Laplacian matrices for undirected graphs. Note that D^+A is right stochastic and thus the matrix of a random walk, hence the name. In the example, it can be observed that each row of the left normalized Laplacian sums to zero. The right normalized Laplacian is used less frequently and sums each column to zero because it is left stochastic. [17]

Adjacency Matrix	Degree Matrix	Left normalized Laplacian	Right normalized Laplacian
$\begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{vmatrix}$	$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & -1 & 0 \\ -1/2 & 1 & -1/2 \\ 0 & -1 & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & -1/2 & 0 \\ -1 & 1 & -1 \\ 0 & -1/2 & 1 \end{vmatrix}$

Figure 10. Example of left and right normalized Laplacian matrices [17]

Considering a non-symmetric adjacency matrix of a directed graph, one must choose between in-degree and out-degree for normalization. Similar to the case of a symmetric adjacency matrix, the left out-degree normalized Laplacian relates to the right stochastic D_{out}^+A, while the right in-degree normalized Laplacian contains the left stochastic AD_{in}^+. Figure 11 demonstrates an example of the out- and in-degree left normalized Laplacian. [17]

Adjacency Matrix	Out-Degree Matrix	Out-Degree left normalized Laplacian	In-Degree Matrix	In-Degree right normalized Laplacian
$\begin{vmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{vmatrix}$	$\begin{vmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & -1/2 & -1/2 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{vmatrix}$	$\begin{vmatrix} 1 & -1 & -1/2 \\ 0 & 1 & -1/2 \\ -1 & 0 & 1 \end{vmatrix}$

Figure 11. Example of left and right normalized Laplacian matrices for directed graphs [17]

2.2 Graph Embeddings

Graph embeddings serve as a feature engineering technique in ML projects. Prior to applying graph embeddings, graphs must be created to provide an efficient data structure to organize data.

Let $G = \{G_1, \dots G_m\}$ be a set of m graphs, and each graph $G_i = \{V_i, E_i, l_i\}_{i=1}^{m}$, where V and E are sets of vertices and edges and l_i the class label of G_i. An embedding is intended as a mapping of a

TU DARMSTADT

whole graph into a low-dimensional space $\mathbb{R}^d$, $d \ll |V|$. The embedding vector should preserve the graph features to the greatest extent possible. [18]

According to NVIDIA, NetworkX is the most popular graph framework used by data scientists who work extensively in the Python environment [19]. NetworkX is designed for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks [20]. This library is constantly updated with extension libraries such as Karate Club[2]. Karate Club is an unsupervised ML extension library from NetworkX that consists of state-of-the-art methods for unsupervised learning on graph structured data [21].

The graph embedding techniques used in this work are all implemented in the Karate Club library which contains not only whole graph embeddings, but also node embeddings and various other methods (see Figure 12). The given whole graph embedding techniques can be divided into three main categories, namely: graph textualization, spectral representation, and statistical representation.

The first category deals with the textualization of graphs, where similar to Doc2vec, which represents a document as a set of words, a single graph is represented by a set of nodes and rooted subgraphs [22]. Graph2Vec is the first whole graph embedding technique from Karate Club. Graph2Vec and its extension, graph and line graph to vector (GL2Vec), are examples for the graph textualization category. The next category is about spectral representation methods like Spectral features (SF), Invariant graph embedding (IGE), and Network Laplacian spectral descriptor (NetLSD) which are more effective for the graph comparison of 3D objects [23]. Since 3D objects have a precise low-dimensional shape, in this category we consider graphs as geometric objects. The last category describes statistical representation methods that generate a graph signature vector based on statistical properties, e.g. median, mean, and standard deviation [23]. Examples of this representation include Family of graph spectral distances (FGSD), Local degree profile (LDP), Feather-Graph, and Geo Scattering. Finally, Wavelet Characteristic is difficult to place into one category because it combines the terms of different categories. Wavelet Characteristic uses both spectral and statistical representation, since its wavelets are constructed by spectral features [24].

[2] https://karateclub.readthedocs.io/en/latest/

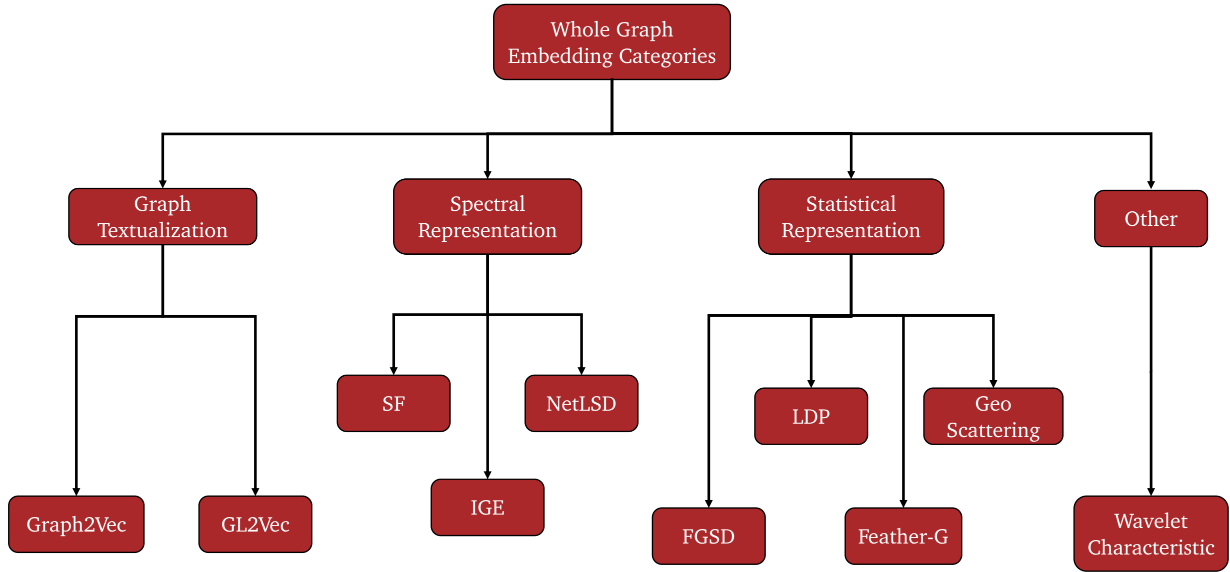


Figure 12. Categorization of Karate Club's whole graph embedding techniques

2.2.1 Graph2Vec

Section 2.2.1 is based on the work developed by [22,25]. Graph2Vec is a data driven representation learning approach for learning whole graph embeddings of arbitrary sizes. The embeddings are learned in an unsupervised manner and can therefore be used for any ML downstream task such as graph classification and graph clustering. When using multiple real-world datasets from the chemo and bio-informatics domain, Graph2Vec outperforms the state-of-the-art graph kernels without losing efficiency. For instance, for malware clustering, the Adjusted Rand Index (ARI) (clustering metric) is 56%, outperforming other state of the art kernels.

2.2.1.1 Notation

Given a set of graphs $\{G_1, G_2, \dots G_n\}$ and an integer λ indicating the size of the embedding, the Graph2Vec algorithm maps the graphs into a set of λ-dimensional graph vectors $\{f(G_1)$, $f(G_2), \dots f(G_n)\}$ called embeddings of G_i. The notion is that Graph2Vec generates embeddings for semantically similar graphs whose values are close to each other.

Let a graph be represented as $G = (V, E, \lambda)$, where V is a set of nodes and $E \subseteq (V \times V)$ a set of edges. Graph2Vec assumes that the Graph G is labeled if there exists a function $\lambda: N \rightarrow \mathcal{L}$ that assigns each node $n \in N$ to a unique label from the alphabet $\mathcal{L}$. When G is unlabeled, then node labeling should be done according to [26] using the node degree.

The whole-graph embedding technique is inspired by Doc2vec of [27], which is a simple extension of Word2Vec that transfers learning from embeddings of words to embeddings of sequences of words. Analogous to Doc2Vec, which represents a single document as a set of words, Graph2Vec represents a single graph as a set of rooted subgraphs.

The algorithm of Graph2Vec starts with a random initialization of the graph embeddings for all graphs in the dataset. Afterwards, rooted subgraphs are extracted around each node in each graph, iteratively learning the corresponding graph embeddings in multiple epochs. In the following, the extraction of rooted subgraphs from a single graph is explained.

2.2.1.2 Extracting Rooted Subgraphs

Before graph embeddings learning is executed, a rooted subgraph $sg_n^{(d)}$ around every node v of G_i is extracted. The extraction of these subgraphs follows the Weisfeiler-Lehman (WL) relabeling strategy. Let d be an integer defining the degree of neighbors to be considered in subgraph extraction. The WL algorithm takes the root node v, the graph G from which the subgraph is to be extracted, and the degree of neighbors d as inputs and returns the intended subgraph $sg_n^{(d)}$. For every node v, Graph2Vec generates $d + 1$ rooted subgraphs with v as the root. The WL relabeling procedure is shown in Figure 13 and explained as following:

1. For each node v in G, locate each label of the nodes adjacent to v and add them to a set of node labels
2. Arrange the elements in ascending order and merge them into one string. Insert the root node label as a prefix into the string
3. The string is mapped to a new ID using some hash function
4. Replace the original label of v with the new generated ID

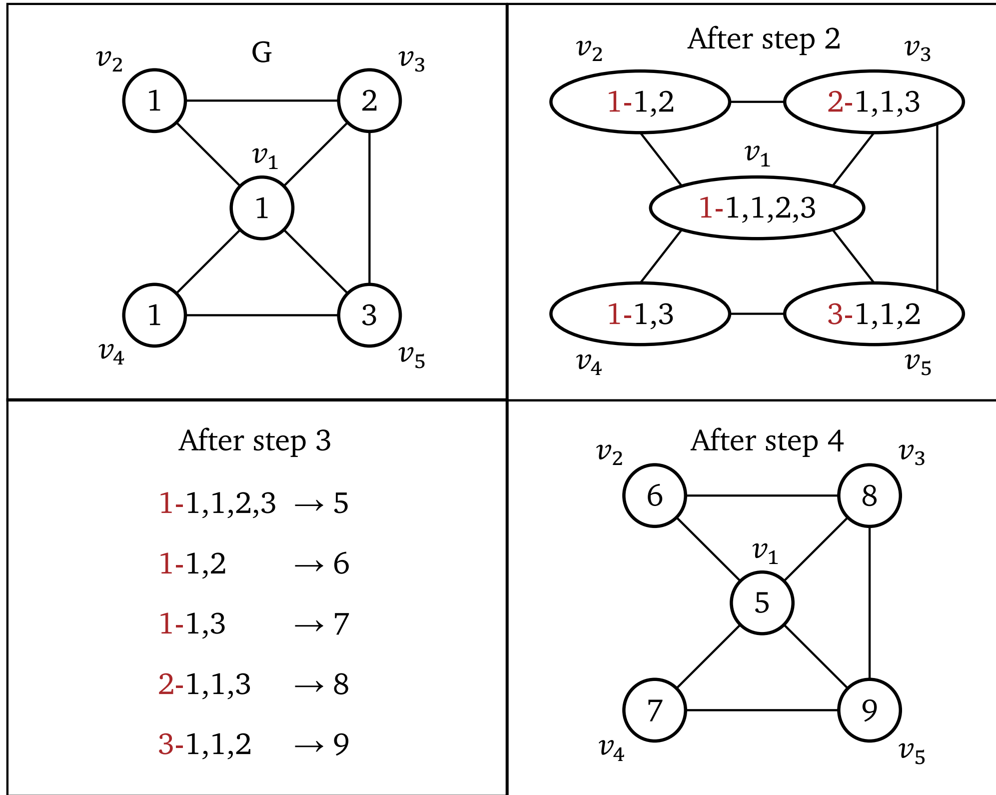


Figure 13. WL relabeling for graph G [25]

To exemplify this, the rooted subgraph of node v_4 is described, for example, as "1-1,3", where "1" is the label of the root and "1,3" describes the labels of the nodes adjacent to v_4. Then the node receives the new subgraph ID "7".

2.2.1.3 Learning Embeddings with Negative Sampling

Let $\{G_1, \dots, G_N\}$ be a set of graphs and $\{c(G_1), \dots, c(G_N)\}$ their subgraphs, then Graph2Vec learns a λ-dimensional embedding $f(G_i)$ for G_i, and also a λ-dimensional embedding for each member subgraph in $c(G_i) = \{sg_1, \dots, sg_n\}$. The model maximizes the log-likelihood specified in Equation (2.3).

$$\sum_{j=1}^{n_i(d+1)} \log P_r\left(sg_j | G_i\right) \tag{2.3}$$

In (2.3) the number of nodes in G_i is n_i and $P_r\ (sg_j|G_i)$ describes the probability that the j-th subgraph sg_j in $c(G_i)$ occurs in G_i. The probability $P_r\ (sg_j|G_i)$ is defined in (2.4).

TU DARMSTADT

$$P_r\left(sg_j \middle| G_i\right) = \frac{\exp\left(f(G_i)\cdot f(sg_j)\right)}{\sum_{sg\in Voc}\exp\left(f(G_i)\cdot f\left(sg_j\right)\right)} \tag{2.4}$$

The problem with calculating this probability is that normally the similarity of all other subgraphs in $c(G_i)$ must be computed for $1 \leq i \leq N$, which is very computationally demanding. Therefore, the skip-gram model can be trained efficiently with negative sampling [28], where just a couple of subgraphs sg are chosen at random and stored in a set called Voc which denotes the randomly chosen vocabulary of subgraphs across all the graphs. After the training converges, the embeddings $f(G_i)$ and $f(G_j)$ are closer if they have similar rooted subgraphs $c(G_i)$ and $c(G_j)$ and at the same time distances them from embeddings which do not have similar subgraphs.

2.2.1.4 Limitations

Since Graph2vec is the first published whole graph embedding of the Karate Club library, there are two major limitations that push for further research. One of them is very straight-forward, Graph2Vec does not consider edge labels. The WL relabeling strategy simply ignores them. The second limitation is related to the quantization of subgraphs of a graph G. When subgraphs are quantized by the WL relabeling procedure, node label information and structural information are both considered. However, the final subgraph IDs for graph G do not always contain sufficient structural information to evaluate the structural similarity between G and other graphs.

2.2.2 Graph and Line Graph to Vector (GL2vec)

Section 2.2.2 is based on the work developed by [25]. GL2Vec was developed as an extension of Graph2Vec that improves on the following two limitations: 1) No consideration of edge labels 2) Graph2Vec does not always preserve enough structural information to evaluate structural similarity. The name reveals that it concatenates the embedding of an original graph with that of the corresponding line graph. In terms of classification, GL2Vec achieves a significant improvement over Graph2Vec. Before describing GL2Vec in detail, the idea of line graphs will be explained below.

2.2.2.1 Line Graphs

Let a graph be represented as $G = (V, E)$, then a so-called line graph $L(G) = (LV, LE)$ represents the adjacencies between the edges of G. $L(G)$ is constructed such that each edge in G is converted to a node in $L(G)$ (see Figure 14). The vertices of $L(G)$ are defined as $LV = \{v(e)|e \in E\}$. Two vertices $v(e_i)$ and $v(e_j)$ in $L(G)$ are connected by an edge if e_i and e_j have a common endpoint in graph G. For instance, since edge (v_1, v_3) and edge (v_1, v_5) have the same endpoint v_1 in G, the line graph $L(G)$ shows an edge between the vertices (v_1, v_3) and (v_1, v_5).

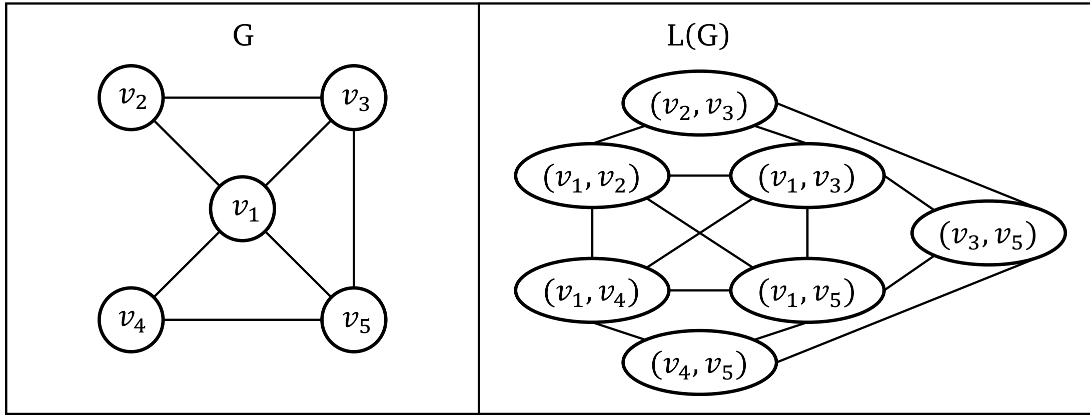


Figure 14. Conversion of graph G to line graph $L(G)$ [25]

As mentioned in Chapter 2.1, the degree of a node v is defined as the number of edges incident to it. Similarly, the number of edges incident to an edge e is declared as the degree of e, symbolized by $\deg(e)$. For an edge e that has the two endpoints v_a and v_b Equation (2.5) holds.

$$\deg(e) = \deg(v_a) + \deg(v_b) - 2 \tag{2.5}$$

2.2.2.2 Method

As mentioned in Chapter 2.2.1 the whole graph embedding technique Graph2Vec does not consider edge labels. Another limitation is that not sufficient structural information is preserved to evaluate the structural similarity between G and other graphs.

Therefore, GL2Vec utilizes the explained line graphs to overcome the limitations of Graph2Vec. In effect, the edges of G become nodes of $L(G)$. The node labels in $L(G)$ are discarded so that the line graph can consider the structural information about G independently of the node labels in G. GL2Vec assumes that graphs have node labels, similar to Graph2Vec. However, it can process graphs with and without edge labels. If the graphs are accompanied by edge labels, the label of an edge e in G is specified as $v(e)$ in $L(G)$. When the graph dataset comes without edge labels, the preservation of structural information is not affected by the node labels of G. But GL2Vec specifies the edge label itself, namely, the degree of edges in G as node labels in $L(G)$.

After reviewing the proposed method GL2Vec, the question arises, what the benefit of using $L(G)$ is if the structural information about G can be extracted from G directly. In this case, the label of a node v in G can be modified to $\deg(v)$. Although this is a possible approach, the line graph $L(G)$ can describe the structure of G more precisely. In fact, there are usually more edges than nodes in G, which means that $L(G)$ can use more values to describe the structure of G. While $L(G)$ uses

degrees and edges, G counts on the degrees of nodes, which allows line graphs to evaluate structural similarity at a finer level.

Line graphs themselves do not consider node labels in G, therefore GL2Vec appends the embedding of $L(G)$ to that of Graph2Vec, in which node labels in G are taken into account. The algorithm of GL2Vec works as follows:

1) Construct line graphs $\{L(G_1), L(G_2), \dots L(G_n)\}$ from the given set of graphs $\{G_1, G_2, \dots G_n\}$. Depending on the availability of edge labels of G, the nodes of $L(G)$ are changed:
   a) Edge labels of G_i are present: each node $v(e)$ in $L(G_i)$ is assigned the label of edge e in G_i
   b) Edge labels of G_i are not present: each node $v(e)$ in $L(G_i)$ is assigned $deg(e)$ in G_i as node label
2) Derive embedding $f(G_i)$ of each G_i by applying Graph2Vec to $\{G_1, G_2, \dots G_n\}$.
3) Derive embedding $g(L(G_i))$ of each $L(G_i)$ by applying Graph2Vec to $\{L(G_1), L(G_2), \dots L(G_n)\}$.
4) Append $f(G_i)$ to $g(L(G_i))$ and the final embedding of G_i is made

2.2.2.3 Limitations

The problem with GL2Vec is that when processing graphs with a very high number of edges, also called dense graphs, the number of vertices increases up to the square of the original graph in its line graph. The creation of line graphs may fail due to a lack of computational resources. It is worth investigating how GL2Vec can be extended to handle such cases. Another opportunity for further research is to find out for what kind of graph datasets GL2Vec works best.

2.2.3 Family of Graph Spectral Distances (FGSD)

Section 2.2.3 is based on the work developed by [29]. Researchers were working extensively on finding a graph feature representation that demonstrates uniqueness, stability, and sparse properties and at the same time is fast to compute. As a result, FGSD and its graph feature representation were developed, which exhibit most of the above properties. For the task of graph classification or graph clustering, it is interesting to know whether two graphs have identical structures. The goal of FGSD is therefore to learn an explicit graph representation that is invariant under graph isomorphism, i.e., under permutation of graph node labels, but also valuable for graph feature extraction.

2.2.3.1 Notion

Given a graph G, we are interested in learning a graph representation, $\mathcal{R}: G \rightarrow (g_1, g_2, \dots, g_r)$, and a feature function $\mathcal{F}: \mathcal{R} \rightarrow (f_1, f_2, \dots, f_d)$ from $\mathcal{R}$ such that the graph features can be used to solve a graph classification problem.

A multiset basically describes a set in which an element can occur multiple times. The proposed method is based on following assumption: an atomic structure of the graph is encoded in the multiset of all pairwise node distances. The pairwise distances of the nodes are generated from an unknown distribution and then connected to form a graph, preserving the pairwise distances. Figure 15 shows the graph generation model based on this assumption.

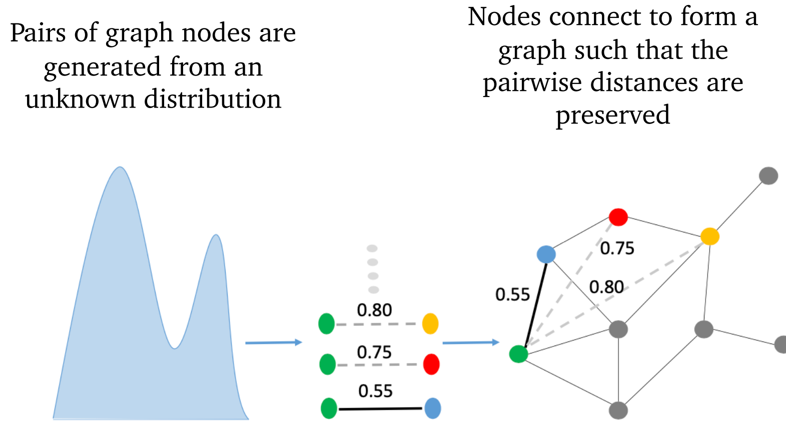


Figure 15. Graph generation model of FGSD [29]

Moreover, the authors show that for a given distance function S_f on a graph, one can explicitly recover all intrinsic properties of the graph while also being able to capture local and global information about the graph. Therefore, $\mathcal{R}$ is defined as the multiset of node pairwise distances that are based on a distance function S_f.

After the discovery of FGSD, researchers have found that harmonic and biharmonic distances on graphs are suitable members of this family for graph representation $\mathcal{R}$. In general, it is important to note that despite of the fact that FGSD does not use node labeling information, it is powerful enough to be competitive on labeled datasets. To solve graph classification problems, we construct a feature vector $\mathcal{F}$ from this histogram of $\mathcal{R}$ and pass it to a classification algorithm.

2.2.3.2 Method

Given a weighted, undirected (and connected) graph $G = (V, E, W)$, where V is a set of vertices, $E \subseteq (V \times V)$ is a set of edges, and W is the nonnegative weighted adjacency matrix. The Laplacian matrix of the graph is defined as $L = D - W$, where D is the degree matrix. The Laplacian matrix is semi-definite, meaning that all its eigenvalues are nonnegative. It allows an eigenvalue decomposition of the form $L = \Phi\Lambda\Phi^T$, where $\Lambda = diag[\lambda_k]$ is the diagonal matrix formed by the eigenvalues, and $\Phi = [\phi_0, \phi_1, \dots, \phi_{n-1}]$ is an orthogonal matrix formed by the corresponding eigenvectors ϕ_k's. For $x, y \in V$, $\phi_k(x)$ and $\phi_k(y)$ are used to describe the x- and y-entry value of ϕ_k. Let f be a nonnegative function on $\mathbb{R}^+$ with $f(0) = 0$, $\mathbf{1} = [1, \dots, 1]^T$ be the all-one vector and

$J = \mathbf{1}\mathbf{1}^T$. With the function f we can define $f(L) := \Phi f(\Lambda)\Phi^T$ and $f(\Lambda) := diag[f(\lambda_k)]$. Finally, L^+ is the Moore-Penrose Pseudoinverse of L.

Given $x, y \in V$, we define the f-spectral distance between x and y on G as follows:

$$S_f(x,y) = \sum_{k=0}^{N-1} f(\lambda_k)\big(\phi_k(x) - \phi_k(y)\big)^2 \quad (2.6)$$

Let $f(\lambda) = \lambda^p$ $(p \geq 1)$, then one can show with (2.6) that

$$S_f(x,y) = (L^p)_{xx} + (L^p)_{yy} - 2\,(L^p)_{xy}. \quad (2.7)$$

$((L^+)^p)_{xy}$ records only p-hop local neighborhood information. Therefore, $((L^+)^p)_{xy} = 0$, when the shortest path from x to y is larger than p. Consequently, for an increasing function of f (e.g., polynomical function with at least $p \geq 1$), S_f captures the *local* structure information.

On the other hand, f as a decreasing function captures the global information specified in (2.8).

$$S_f(x,y) = ((L^+)^p)_{xx} + ((L^+)^p)_{yy} - 2\,((L^+)^p)_{xy}. \quad (2.8)$$

Many known graph distances can be derived from the FGSD sub-family. The harmonic distance for $f(\lambda) = \frac{1}{\lambda}$ where $\lambda > 0$ is $S_f(x,y)$. The polyharmonic distance for $f(\lambda) = \frac{1}{\lambda^p}$ where $p \geq 1$ is $S_f(x,y)$. The biharmonic distance is a special case of this function with $p = 2$. Finally, the heat diffusion distance uses $f(\lambda_k) = e^{-2t\lambda_k}$ for $S_f(x,y)$.

2.2.3.3 Computation

FGSD has a worst-case complexity of $O(N^2)$ when an exact calculation of $\mathcal{R}$ is conducted. However, an approximation can also be performed that yields a complexity of $O(r|E|)$, where $|E|$ is the number of edges and r is the number of approximations.

2.2.4 Network Laplacian Spectral Descriptor (NetLSD)

Section 2.2.4 is based on the work developed by [12]. In graph analytics, it is hard to find a graph comparison model that employs an expressive similarity measure and at the same time is

computationally efficient. From an ideal point of view, graph comparison should be invariant to the order of nodes and the sizes of the compared graphs, adapt to the scale of the graph patterns and be scalable. Until the publication of NetLSD, these properties have not been addressed together, instead graph comparison was performed with graph kernels like in 2.2.1 or 2.2.2 and statistical representation-based methods (see 2.2.3), which are inefficient for large graph collections. Therefore, researchers proposed NetLSD, which is a permutation- and size-invariant, scale-adaptive, and efficiently computable graph representation for the comparison of large graphs. It extracts a dense signature that adopts the formal properties of the Laplacian spectrum, in particular the heat or wave kernel. The authors state that NetLSD “hears the shape of a graph”.

2.2.4.1 Problem Statement

Given an undirected graph $G = (V, E)$, where V is the set of vertices, and $E \subseteq (V \times V)$ the set of edges. It is assumed that the graph is unweighted, although NetLSD can also apply the weighted case. A representation is a function $\sigma: \mathcal{G} \rightarrow \mathbb{R}^{\mathbb{N}}$ from any graph G in a set of graphs $\mathcal{G}$ to an indefinite dimensional real vector, where element j of the representation is denoted as $\sigma_j(G)$. A representation-based distance is a function $d^{\sigma}: \mathbb{R}^{\mathbb{N}} \times \mathbb{R}^{\mathbb{N}} \rightarrow \mathbb{R}_0^+$ on the representation of two graphs $G_1, G_2 \in \mathcal{G}$ that returns a positive real number. The goal is to develop a time constant based distance between an arbitrary pair of graphs G_1, G_2.

2.2.4.2 Expressive Graph Comparison

To offer permutation-invariant, scale-adaptive, and size-invariant graph comparison, expressive distances are required. Permutation invariance means that if two graphs are isomorphic, the distance between their representations is zero.

The distance d^{σ} on the representation σ is permutation-invariant if:

$$\forall G_1, G_2 \;\; G_1 \simeq G_2 \;\Rightarrow d^{\sigma}\big(\sigma(G_1), \sigma(G_2)\big) = 0$$

Scale-adaptivity implies that a representation describes local and global graph features. It is defined as a property of a representation σ having minimum one local feature (which is derived only from information encoded in subgraphs $\xi(G)$), and minimum one global feature (which is derived by more than the information encoded in any $\xi(G)$). A representation is scale adaptive if it accounts for both local features σ_i and global features σ_j:

- Local Feature: $\forall G \; \exists \; f(\cdot): \sigma_i = f(\xi(G))$
- Global Feature: $\forall G \; \nexists \; f(\cdot): \sigma_j = f\big(\xi(G)\big)$

Size-invariance is the ability to identify that two graphs reflect the same phenomena at different sizes. For instance, two criminal circles with similar structure but different sizes should have a distance close to zero. This property requires that the two graphs originate from the sampling of the same domain $\mathcal{M}$. A size-invariant distance d^{σ} on the representation σ meets this condition:

$$\forall \mathcal{M}: G_1, G_2 \text{ sampled from } \mathcal{M} \Rightarrow d^{\sigma}\big(\sigma(G_1), \sigma(G_2)\big) = 0$$

2.2.4.3 Network Laplacian Spectral Descriptor

In general, it is difficult to find a representation that satisfies the above requirements, so NetLSD proposes to transfer the problem to the spectral domain. To do so, two applications from the classic physics were used. In the first method, the nodes of the graph are heated, and the heat diffusion is observed over time. In the second method, a system of masses corresponds to the nodes of the graph and springs correspond to its edges, where the propagation of mechanical waves is detected. Our representation uses a trace signature that embodies such a heat diffusion or wave propagation process over time. The comparison between two graphs is made using the L_2 distance between trace signatures recorded on selected time scales.

Given the adjacency matrix A of a graph G, the normalized Laplacian matrix is defined as $\mathcal{L} = I - D^{-1/2}AD^{-1/2}$, where D is the diagonal matrix. The set of eigenvalues of $\mathcal{L}$ is called the *spectrum* of a graph. The spectrum of the normalized Laplacian is bounded between $0 \le \lambda_i \le 2$. Instead of using the Laplacian spectrum directly, NetLSD considers a heat diffusion process on the graph to obtain an expressive representation similar to random walk models.

The closed-form solution of the heat equation provides the heat at each vertex at time t and is given by the heat kernel matrix in (2.9).

$$H_t = e^{-t\mathcal{L}} = \sum_{j=1}^{n} e^{-t\lambda_j} \cdot \phi_j \phi_j^T \tag{2.9}$$

$(H_t)_{ij}$ is the amount of heat transferred from vertex v_i to vertex v_j at time t. Since the heat kernel matrix contains nodes, it is not suitable for graph comparison. Therefore, the heat trace at time t

$$h_t = tr(H_t) = \sum_{j} e^{-t\lambda_j} \tag{2.10}$$

is a better representation. Finally, the representation of NetLSD consists of a collection of heat traces at different time scales $h(G) = \{h_t\}_{t>0}$. From a physical point of view, the heat kernel can be considered as a family of low-pass filters, and therefore the heat trace signature contains low-frequency information at each scale.

Alternatively, a solution of the wave equation describes a wave propagation in a medium, and a solution is given by the wave kernel in (2.11).

$$W_t = e^{-it\mathcal{L}} = \sum_{j=1}^{n} e^{-it\lambda_j} \cdot \phi_j \phi_j^T \tag{2.11}$$

Using (2.11) we can calculate the corresponding wave trace signature with $t \in [0, 2\pi)$ as follows:

$$w_t = tr(W_t) = \sum_j e^{-it\lambda_j} \tag{2.12}$$

Finally, the proposed framework of NetLSD outperforms FGSD on a variety of graph collections on community detection and graph classification. Furthermore, the complexity of NetLSD is more efficient than FGSD with $\mathcal{O}(km + k^2 n)$, where k represent the eigenvalues of the normalized Laplacian, m the number of edges, and n the number of nodes.

2.2.5 Spectral Features (SF)

Section 2.2.5 is based on the work developed by [30]. In graph mining, many approaches from different areas of ML are used to classify graphs. Most of these approaches (e.g., kernel models, sequential models, etc.) are based on complex mathematical methods and require high computational power. However, SF is a simple and fast algorithm that relies on the spectral decompensation of the graph's Laplacian matrix to perform graph classification.

Given an undirected and unweighted graph $G = (V, E)$ and its corresponding boolean adjacency matrix $A \in \{0,1\}^{|V| \times |V|}$. It is assumed that G is connected, if not, the largest connected component of G is extracted. Let D be the node degree matrix, then the normalized Laplacian matrix can be calculated with (2.13).

$$\mathcal{L} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{2.13}$$

For the graph embedding X, the k smallest positive eigenvalues of $\mathcal{L}$ are used in ascending order:

$$X = (\sigma_1, \dots, \sigma_k)$$

When the graph has less than k nodes, the remaining nodes are filled with zeros to obtain a vector of the corresponding dimension: $X = (\sigma_1, \dots, \sigma_{|V|-1}, 0, \dots, 0)$. The embedding X is defined as spectral features. A major advantage of the spectrum representation is that it is independent of the labeling of the nodes. Then, the k lowest eigenvalues are used as input to a classifier, and the predicted class $\hat{c}$ is the output of the SF model (see Figure 16).

The eigenvalues of the normalized Laplacian matrix $\mathcal{L}$ have a bounded spectrum $0 \leq \lambda_i \leq 2$. This is a beneficial property for the classifier since no extensive rescaling or preprocessing needs to be done. It can be observed that X does not include σ_0. The reason for this is that the eigenvalue 0 provides information about the number of connected components in the graph, but in this representation only the largest connected components are considered. Note also that other values indicate the presence of certain structures in the graph, e.g., the eigenvalue 2 implies a bipartite structure.

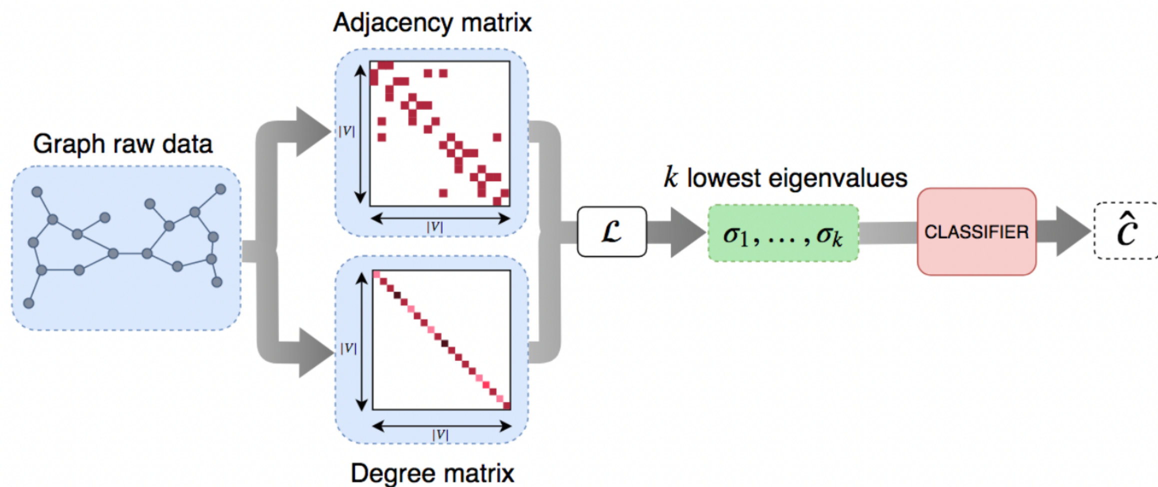


Figure 16. Schematic illustration of the SF model. $\mathcal{L}$ denotes the normalized Laplacian and $\hat{c}$ the predicted class. [30]

There are some physical interpretations for the eigenvalues of the Laplacian. For example, in [31] it is described that the eigenvalues correspond to the frequencies associated with a Fourier decomposition of any signal on the vertices of the graph. Since only the smallest eigenvalues are chosen as input to the classifier, the shear of the Fourier decomposition acts as a low-pass filter on the signal. Metaphorically speaking, the characterization of the graph by the smallest eigenvalues of $\mathcal{L}$ can be compared to the characterization of a melody by its lowest fundamental frequencies.

Other graph embedding techniques such as 2.2.4 and 2.2.6 propose methods for combining spectral decomposition and graph isomorphism, but this is not addressed in SF. Instead, SF proposes a simple and fast algorithm to obtain an initial reference score for a dataset.

2.2.6 Invariant Graph Embedding (IGE)

Section 2.2.6 is based on the work developed by [32]. In graph learning, researchers seek graph representations that are able to distinguish between different graphs while maintaining low computational complexity. When the graph representation is permutation-invariant, it should be both invariant to node permutation and able to distinguish two non-isomorphic graphs. However, IGE suggests a graph representation that relies on spectral graph theory and is still invariant for a large family of graphs. As the name implies, the graph embedding is invariant, but two graphs that are not identical can produce an identical embedding $\mathcal{F}_{\text{IGE}}(G)$.

2.2.6.1 Notation

Given an undirected graph without self-loops $G = (V, E)$, where V is a set of nodes and $E \subseteq (V \times V)$ is a set of edges. The number of nodes is equal to the size of the graph and is denoted by $n = |V|$. It has a boolean adjacency matrix $A \in \{0,1\}^{|V|\times|V|}$ if there are no edge weights. In the presence of edge weights, A_{ij} is the weight between the nodes i and j. Let D be the diagonal matrix of the node degrees, then the Laplacian matrix is defined as $L = D - A$. The transition matrix for the random walk on graph G is defined as $P = D^{-1}A$.

Recall that a graph embedding is a function $\mathcal{F}$ mapping graphs to vectors in $\mathbb{R}^d$, where d is the dimension of the embedding. Given two isomorphic graphs G and H, a graph embedding is invariant if $\mathcal{F}(G) = \mathcal{F}(H)$.

A graph embedding can be easily created by embeddings of nodes, but a way must be found to deal with varying graph sizes. If the embedding of nodes $\mathcal{E}$ is equivariant[3], applying any symmetric function to it can produce an invariant graph embedding $\mathcal{F}$.

The IGE consists of three separate embeddings, which are explained in more detail below:

2.2.6.2 Invariant Graph Embedding from Eigenvalues

It is assumed that the graph G is connected and its Laplacian has positive semi-definite eigenvalues $\lambda_1 = 0 < \lambda_2 \leq \lambda_3 \leq \cdots \leq \lambda_n$. For a fixed parameter k_1 and for the case that $n \geq k_1 + 1$ holds, the embedding is defined as $\mathcal{F}_1(G) = (\lambda_2, \dots, \lambda_{k_1+1})$, but if $n \leq k_1$, then $\mathcal{F}_1(G) = (0, \dots, 0, \lambda_2, \dots, \lambda_n)$, completing this vector up to size k_1. Since the spectral representation does not depend on the

[3] functions whose values are unchanged by a symmetric transformation [58]

indexing of the nodes, it is obvious that this embedding is invariant. However, there are still non-isomorphic graphs that have the same eigenvalues.

2.2.6.3 Space Embedding

Recall that P_{ij}^k is the probability for a random walk started in node i to be in node j after k steps. When node features F are given, then $P^k F = (x_1^k, \dots, x_n^k)$ is the aggregation of the features of its k-hop neighbors. Apparently, $P^k F$ is an equivariant embedding such that for any symmetric function $f: \mathbb{R}^n \to \mathbb{R}^m, f(x^k(1), \dots, x^k(n))$ is an invariant feature of the graph. If no features F are given, we use $F = (d_1, \dots, d_n)$. For the symmetric function of IGE we define

$$f(x_1, \dots, x_n) = hist(x_1, \dots, x_n; t),$$

where $hist$ is the histogram function and t is the number of bins of the histogram. To sum up, the second embedding $\mathcal{F}_2(G) \in \mathbb{R}^{k_2 t_2}$ is obtained by connecting the features defined above with the number of bins t_2 and apply it to the vector $P^k F$ for values of $k \in \{1, \dots, k_2\}$ for a fixed parameter k_2.

2.2.6.4 Invariant Graph Embedding from Commuting

Recall that the Laplacian matrix L allows an eigenvalue decomposition of the form $L = \Phi \Lambda \Phi^T$, where $\Lambda = diag(\lambda_1, \dots, \lambda_n)$ is the diagonal matrix of the eigenvalues of L, and $\Phi = (\phi_1, \dots, \phi_n)$ is the matrix of corresponding eigenvectors, with $\Phi^T \Phi = I$ and $\phi_1 = 1/\sqrt{n}$. With this information, we define a classical spectral embedding for nodes $X = \sqrt{\Lambda^+} \Phi^T$, where $\Lambda^+ = diag(0, 1/\lambda_2, \dots, 1/\lambda_n)$ denotes the pseudo-inverse of Λ. Each column of $x(1), \dots, x(n)$ defines an embedding of the nodes in $\mathbb{R}^n$.

To derive the embedding, we give it a random walk interpretation, where a random walk with transition rate A_{ij} from node i to j is considered. The walker remains at node i for an exponential time with parameter d_i and then walks from node i to node j with probability $P_{ij} = A_{ij}/d_i$. The sequence of visited nodes, forms a discrete Markov chain with transition matrix P. Let H_{ij} be the mean hitting time of node j from node i. Then the mean commuting time between node i and j is defined in (2.14).

$$C_{ij} = H_{ij} + H_{ji} = n\|x(i) - x(j)\|^2 \quad (2.14)$$

Since the geometry of the node embeddings $x(i)$ is related to the geometry of the graph, the matrix of commuting times characterizes the graph. For example, the adjacency matrix can only be reconstructed with knowledge of the matrix of commuting times.

C_{ij} contains all the information about the graph, so it is included in the embedding. However, instead of the Euclidean distances $\|x(i) - x(j)\|^2$, the dot product $x(i)^T x(j)$ is calculated for $1 \leq j$, $j \leq n$. Then, the matrix is converted into a vector and passed through a histogram. The embedding is defined as follows:

$$\mathcal{F}_3(G) = hist(\big(x(i)^T x(k)\big)_{1\leq i,k\leq n}; t_3) \in \mathbb{R}^{t_3} \tag{2.15}$$

$\mathcal{F}_3(G)$ is obviously an invariant embedding of the graph but cannot reconstruct the graph.

2.2.6.5 Invariant Graph Embedding

The final embedding can be constructed by concatenating all three embeddings for fixed parameters k_1, k_2 and t_2, t_3: $\mathcal{F}_{\mathrm{IGE}}(G) = (\mathcal{F}_1(G),\ \mathcal{F}_2(G),\ \mathcal{F}_3(G))$. Each of these embeddings is invariant, but two different graphs can produce the same embedding $\mathcal{F}_{\mathrm{IGE}}$. The search for non-isomorphic graphs where this embedding is not discriminative is not covered in IGE.

2.2.7 Local Degree Profile (LDP)

Section 2.2.7 is based on the work developed by [33]. The field of representation learning on graphs is becoming larger and recently more algorithms based on graph kernels and graph neural networks were developed especially for graph classification. LDP proposes a simple statistical graph representation with linear time complexity that outperforms similar state-of-the-art graph kernels and graph neural networks for non-attributed graph classification. Although LDP does not incorporate attributes, its performance on classifying attributed graphs is slightly weaker but it still serves as an effective method for attributed graph classification.

Given a graph $G = (V, E)$ where V is the set of nodes and E is a set of edges, let $DN(v)$ be the multiset of the degree of all the neighboring nodes of v, denoted as $DN(v) = \{degree(u) | (u, v) \in E\}$. LDP proposes to take five node features, which are

$$\big(degree(v), \min\big(DN(v)\big), \max\big(DN(v)\big), \mathrm{mean}\big(DN(v)\big), \mathrm{std}\big(DN(v)\big)\big).$$

These features capture information about each node and its first neighborhood. Then all node features of a graph are mapped into a histogram or an empirical distribution.

LDP stands out due to its low computational complexity. In feature extraction, the degree of each node is counted, and the statistics of the first neighborhood is stored for each node with a time complexity of $\mathcal{O}(E)$. Subsequently, the V numbers need to be mapped into B bins, which takes $\mathcal{O}(V)$ time, so the overall time complexity of LDP is $\mathcal{O}(E)$.

The LDP model has similarities with the WL kernel and thus with 2.2.1 and 2.2.2. Both algorithms start from local node features and build new features from the previous step through the graph structure. However, the LDP model does not include the hashing step of the WL kernel, which relabels the nodes and possibly loses information about local similarities, i.e., two nodes with very similar neighborhoods could receive completely different labels.

2.2.8 Geo Scattering

The field of computer vision is a good example of the advantages of deep neural networks. Analogous to the generalization of convolutional networks in geometric deep learning, Geo Scattering explores the generalization of scattering transforms from normal signals to graph data.

2.2.8.1 Wavelets

Before understanding the concept of geometric scattering, we first have to introduce the notion of a wavelet and of wavelet scattering.

Wavelets are wave-like oscillations that begin and end at zero and have an average value of zero. They have limited duration and tend to be irregular and asymmetric. For example, Figure 17 illustrates Daubechies wavelets. [34]

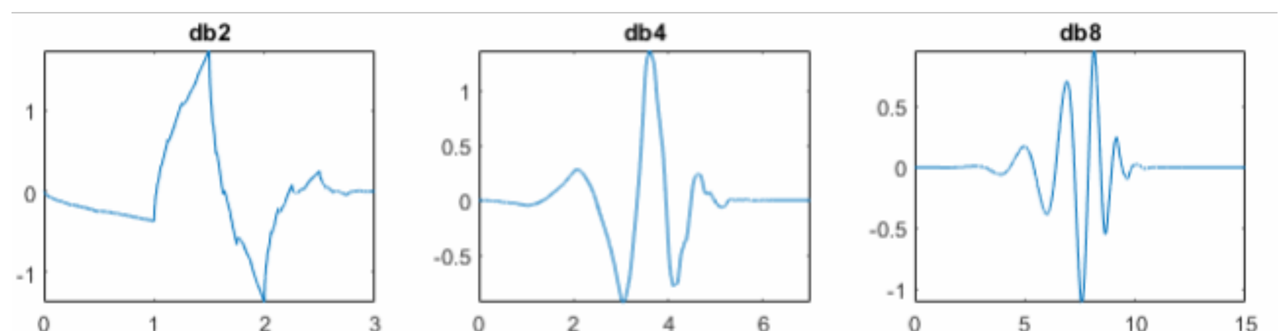


Figure 17. Example of Daubechies wavelets with N = 2, 4, 8 [35]

Many recorded signals from natural processes are characterized by rapid ups and downs. Unlike in Fourier analysis, where the basis consists of harmonic sine waves, wavelets are able to display piecewise regular signals and images that exhibit transient behavior sparsely. While Fourier

analysis consists of breaking up a signal into sine waves with different frequencies, wavelet analysis breaks a signal into shifted and scaled versions of the mother wavelet. In general, wavelet analysis reveals features of a signal or image that other analysis techniques miss, such as trends, collapse points or unsteadiness in higher derivatives. [34]

2.2.8.2 Wavelet Scattering Transform

With the knowledge of a wavelet, which can be defined as a function $\Psi(t)$ over time, we have to distinguish between the regular Euclidean wavelet scattering and the proposed geometric wavelet scattering of [36]. Euclidean wavelet scattering allows the inference of low-variance features from time series and image data for use in ML applications. The scattering network utilizes predefined wavelet and scaling filters. [37]

The wavelet scattering transform leads to the extraction of the desired scattering coefficients. The transformation process consists of different stages, where the output of one stage becomes the input for the next stage. Starting from a feature vector x as input, the wavelets Ψ and the scaling function ϕ, Figure 18 demonstrates the operations that are performed in each stage for a feature vector. [37]

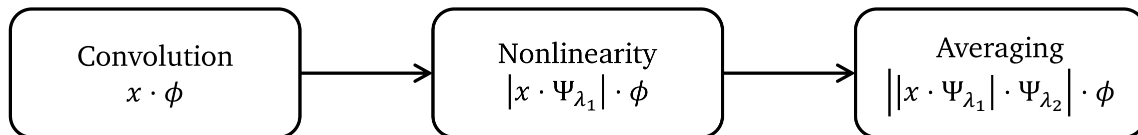


Figure 18. Operations of the wavelet scattering transform inspired by [37]

It should be noted that Euclidean scattering transform is designed for wavelets defined on $\mathbb{R}^d$. Geo Scattering extends the construction to graphs. Hence, graph wavelets are defined as the difference of lazy random walks spread over different time scales, where the foundation of these graph wavelets lies in the properties of the graph Laplacian. [36]

2.2.8.3 Geometric Scattering

Sections 2.2.8.3 and 2.2.8.4 are based on the work developed by [36]. The architecture for obtaining graph embeddings through geometric scattering is illustrated in Figure 19.

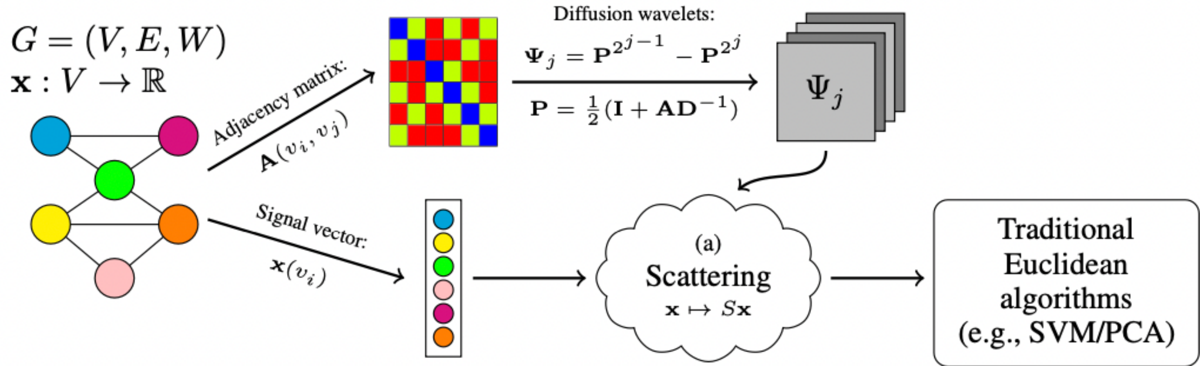


Figure 19. Architecture for using Geo Scattering of graph G and signal x [36]

Given a weighted graph $G = (V, E, A)$, where V is the set of vertices, $E \subseteq (V \times V)$ the set of edges, and A the adjacency matrix containing the weights of the edges. Then we define $\mathrm{P} = \frac{1}{2}(\mathrm{I} + \mathrm{AD}^{-1})$ as the lazy random walk matrix, where the sum of all column entries equals one. The matrix P^{t} determines the probability distribution of a lazy random walk after t steps. A random walk is a special case of a walk in which the steps $v_{l_0}, v_{l_1}, \dots$ are chosen randomly and allow $v_{l_i} = v_{l_{i+1}}$. The corresponding wavelet matrix at the scale 2^j is defined in (2.17).

$$\Psi_j = \mathrm{P}^{2^{j-1}} - \mathrm{P}^{2^j} = \mathrm{P}^{2^{j-1}}\left(\mathrm{I} - \mathrm{P}^{2^{j-1}}\right) \tag{2.16}$$

Since Euclidean scattering transform conducts three operations to determine the scattering coefficients, the zero, first, and second order scattering moments are defined in the following. For this purpose, it is important to know that a moment of a function is a quantitative measure related to the shape of the function's graph. When the function is a probability distribution, then the normalized (also standardized) moments are its mean, variance, skew, and kurtosis. The not normalized q^{th} moments of x returns

$$S\mathrm{x}(q) = \sum_{l=1}^{n} \mathrm{x}(v_l)^q \quad 1 \leq q \leq Q, \tag{2.17}$$

as zero order scattering moment. Similar to that we can extract invariant coefficients from $|\Psi_j \mathrm{x}|$ by computing its moments

TU DARMSTADT

$$Sx(j,q) = \sum_{l=1}^{n} |\Psi_j x(v_l)|^q \quad 1 \leq j \leq J, 1 \leq q \leq Q, \tag{2.18}$$

which are defined as first order geometric moments. These coefficients allow a finer separation of the frequency responses of x. More specifically, while $Sx(2)$ mixes all frequencies of x, it can be observed that $Sx(j,2)$ mixes only the frequencies of x that are captured by Ψ_j. The first order geometric moments can be extended by iterating both the graph wavelet and absolute value transforms. These moments are defined as second order geometric moments and can be determined with:

$$Sx(j,j',q) = \sum_{l=1}^{n} |\Psi_{j'}|\Psi_j x(v_l)||^q, 1 \leq j \leq j' \leq J, 1 \leq q \leq Q \tag{2.19}$$

The invariant statistics $Sx(j,j',q)$ consist of applying the wavelet transform operator $\Psi^{(J)}$ to each $|\Psi_j x|$ and calculating the summary statistics of the magnitudes of the resulting coefficients. Second order moments combine two scales 2^j and $2^{j'}$ within graph G, creating features that combine patterns from smaller sub-graphs within G with patterns from larger sub-graphs. For example, individuals in friend circles and larger community structures in social network graphs.

Finally, the collection of zero, first, and second order geometric scattering moments $Sx = \{Sx(q), Sx(j,q), Sx(j,j',q)\}$ offers an extensive set of invariants of the graph G.

2.2.8.4 Conclusion

Geometric scattering transform is an approach for feature extraction on graphs that generalizes the Euclidean scattering transform. It provides a new way of computing graph embeddings because it is independent of specific learning tasks and can be used for both supervised and unsupervised applications. Although it is not clear whether a scattering model is suitable for graph classification, the results of the evaluation with different datasets revealed that this method has potential and can serve as universal representation of graphs.

2.2.9 Feather-Graph

Section 2.2.9 is based on the work developed by [38]. The following approach deals with characteristic functions defined on graph vertices to describe the distribution of vertex attributes at multiple scales. Feather-Graph is an algorithm to calculate a specific variant of these characteristic functions on large, attributed graphs in linear time. The specific variant is called the

r-scale random walk weighted characteristic function, which is discussed in more detail in this section. We consider a specific case where the probability weights of the characteristic function are defined as the transition probabilities of random walks. This method extracts node-level features, which are then pooled to create compact descriptors of graphs for graph classification tasks. Feather-Graph is robust against data corruption and isomorphic graphs have the same vector space representation. The implementation of Feather-Graph has only been tested for graph classification with datasets involving social media networks and web pages.

2.2.9.1 Notion of Characteristic Functions

Network datasets can contain multiple attributes that affect the characteristics of a node and the network, but these neighborhood features are complex to interpret because they can have an unlimited range with unknown distributions. To utilize neighborhood information, Feather-Graph uses characteristic functions because, regardless of the type of distribution, there is always a unique characteristic function that can be combined across multiple nodes and even multiple attributes. This allows us to compare different neighborhoods uniformly.

2.2.9.2 Characteristic Functions on Graphs

This section addresses the idea of describing the distributions of node features by characteristic functions. Feather-Graph proposes the use of r-scale random walk weighted characteristic functions, which are computed below for all nodes in linear time.

Let $G = (V, E)$ be an attributed and unweighted graph, where the nodes of G have the random variable X also defined as the feature vector $\mathrm{x} \in \mathbb{R}^{|V|}$, where x_v is the feature value for node $v \in V$. The objective is to describe the distribution of this feature in the neighborhood of $u \in V$. We can define the characteristic function of X with the imaginary unit i for the source node u at the evaluation point of the characteristic function $\theta \in \mathbb{R}$ with

$$\mathrm{E}\left[e^{i\theta X}|G, u\right] = \sum_{w \in V} P(w|u) \cdot e^{i\theta \mathbf{x}_w}, \tag{2.20}$$

where the affiliation probability $P(w|u)$ describes the strength of the relationship between the source node u and the target node w. The characteristic function can be divided into the real and imaginary part by using Euler's identity as follows:

$$\mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X}|G, u\right]\right) = \sum_{w \in V} P(w|u)\cos(\theta \mathrm{x}_w) \tag{2.21}$$

$$\text{Im}\left(\text{E}\left[e^{i\theta X}|G,u\right]\right) = \sum_{w\in V} P(w|u)\sin(\theta \text{x}_w) \quad (2.22)$$

It can be observed that the real and imaginary units are sums of sine and cosine waves where the amplitude is $P(w|u)$, the evaluation point θ is equivalent to time and the feature vector x_w describes the angular frequency.

After the defining the general characteristic function, we can now specify the affiliation probability $P(w|u)$ between the source node u and the target node w. Recall that the sequence of nodes in a random walk of G is $\{v_j, v_{j+1}, \dots, v_{j+r}\}$. Let r be the scale of the neighborhood of node u, then every node in the neighborhood of u can be reached in r steps by a random walk from source node u. We can describe the distribution of the features in the neighborhood u with

$$\text{Re}\left(\text{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} P\left(v_{j+r} = w \middle| v_j = u\right)\cos(\theta \text{x}_w) \quad (2.23)$$

$$\text{Im}\left(\text{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} P\left(v_{j+r} = w \middle| v_j = u\right)\sin(\theta \text{x}_w), \quad (2.24)$$

where $P\left(v_{j+r} = w \middle| v_j = u\right)$ is the probability of a random walk starting in source node u reaching the target node w within r steps. Let A be the adjacency matrix of G and D its corresponding degree matrix, then the normalized adjacency matrix can be defined as $\widehat{\text{A}} = \text{D}^{-1}\,\text{A}$. This normalized adjacency matrix can be used with its rth power to describe the probability of a source-target node pair (u, w) in a neighborhood with scale r as $\widehat{\text{A}}^r_{u,w} = P\left(v_{j+r} = w \middle| v_j = u\right)$. Integrating $\widehat{\text{A}}^r_{u,w}$ in (2.23) and (2.24) results in:

$$\text{Re}\left(\text{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} \widehat{\text{A}}^r_{u,w}\cos(\theta \text{x}_w) \quad (2.25)$$

$$\text{Im}\left(\text{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} \widehat{\text{A}}^r_{u,w}\sin(\theta \text{x}_w) \quad (2.26)$$

Figure 20 illustrates the r-scale random walk weighted characteristic function of the logarithmically transformed degree as feature vector, for a high and low degree node in the Twitch England Network [39]. It can be observed that the real part contains even functions, and the imaginary part contains odd functions. The range of the y-scale values is $[-1,1]$ and nodes with different structural degree have different functions.

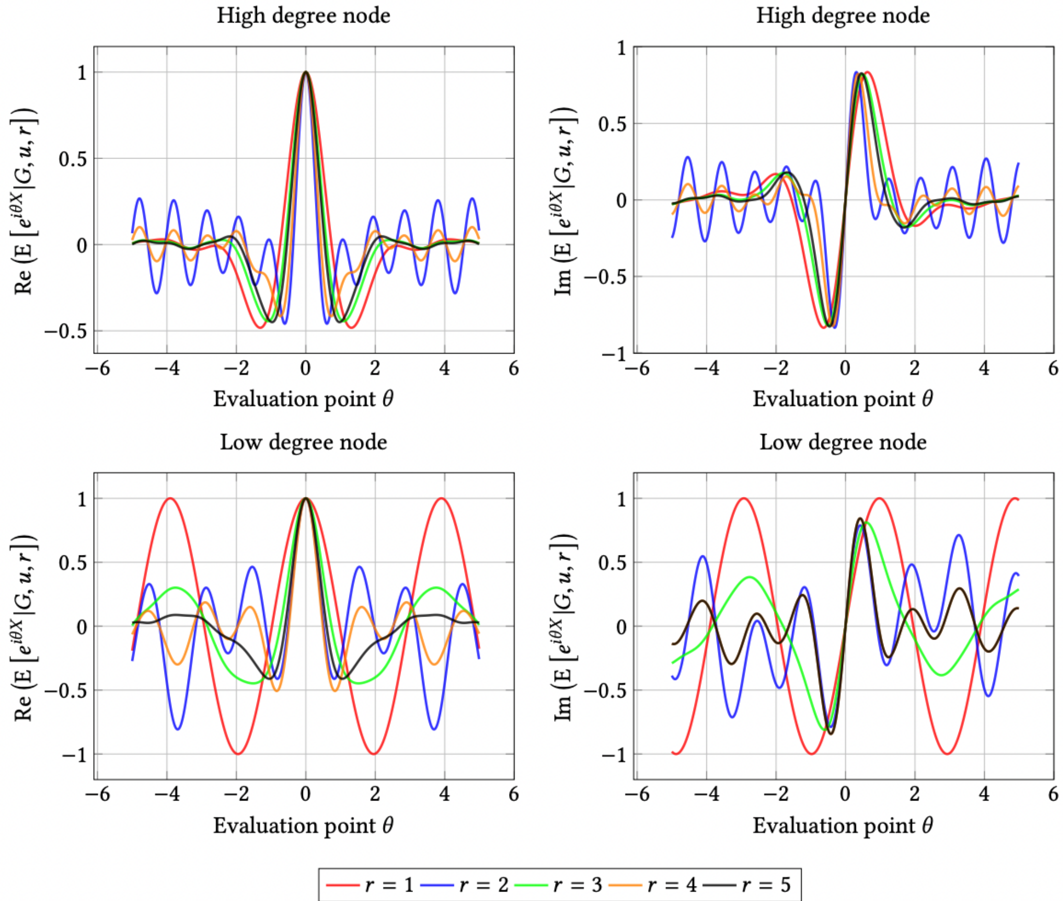


Figure 20. r-scale random walk weighted characteristic functions for a low and high degree node [38]

2.2.10 Wavelet Characteristic

Section 2.2.10 is based on the work developed by [24]. Wavelet Characteristic is an unsupervised whole graph embedding approach that uses spectral graph wavelets to capture topological similarities in each k-hop sub-graph between nodes. The extracted similarities are used to learn embeddings for the whole graph. Wavelet Characteristic produces identical embeddings for isomorphic graphs and is resistant to feature noise. The proposed method outperformed seven of the explained graph embedding techniques in 2.2 and was published in 2021 as the most recent. The evaluation of the method was performed for graph classification on social media datasets.

The procedure for obtaining the embedding consists of two parts: 1) computing the topological wavelet similarity and (2) characterizing the distribution of features in subgraphs. In the first part, we calculate the topological similarity of nodes based on diffusion wavelets and use them to collect the distribution of node features in sub-graphs. After obtaining the characteristic functions of k-hop sub-graphs, representative sample points were selected to concatenate the results and obtain the graph-level embedding.

2.2.10.1 Topological Wavelet Similarity

Let L be the Laplacian matrix and $\lambda_1 \leq \lambda_2 \leq \lambda_N$ its eigenvalues, then L can be decomposed as $L = U\Lambda U^T$, $\Lambda = diag(\lambda_1, \dots, \lambda_N)$. The eigenvalues of L can be seen as temporary frequencies of a signal on the graph G. To suppress larger eigenvalues and smooth the signal, the filter kernel g_τ with its variable paramter τ is used, which is here the heat kernel $g_\tau = e^{-\lambda\tau}$. With the knowledge of the heat kernel, we can calculate the spectral wavelet coefficient matrix Ψ with (2.27).

$$\Psi = U\ diag\big(g_\tau(\lambda_1), \dots, g_\tau(\lambda_N)\big)\ U^T \tag{2.27}$$

The element Ψ_{ji} indicates for a given node v_i how much energy is transported from node v_j to v_i. The i-th column of the matrix describes a distribution of energy from the other nodes to v_i. According to [40] the energy wavelet distribution of two nodes represents their topological distance.

Finally, to determine the topological similarity between different nodes, we need to look at the minimum difference of pair assignments (MDPA). The MDPA can quickly determine the difference between two histograms and can be calculated in linear time under certain conditions. Let X and Y be two sets of n elements, then the MDPA between both sets is defined as $\sum_{i=1}^{n} |x_i - y_i|$. To determine the MDPA between Ψ_i and Ψ_j, we need to arrange both vectors in ascending order to calculate the pairwise distance. We can define the topological wavelet similarity of two nodes v_i and v_j as follows:

$$s(v_i, v_j) = e^{-MDPA(\Psi_i, \Psi_j)} \tag{2.28}$$

2.2.10.2 Sub-graph Feature Distribution

Given A as attribute matrix and $\hat{a}_i$ as vector with the features of node v_i. Let $G_k(v_i)$ be the feature distribution in k-hop sub-graph $G_k(v_i)$, then the characteristic function of $\hat{a}_i$ in $G_k(v_i)$ is

$$\phi_{v_i}^{(k)}(t) = \mathbb{E}\left[e^{it\hat{a}_i} \mid G_k(v_i)\right] = \sum_{v_j \in G_k(v_i)} P(v_j|v_i) e^{ita_j}, \tag{2.29}$$

where the transition probability $P(v_j|v_i)$ is proportional to the similarity between nodes v_j and v_i and the influence of node v_i. To calculate the characteristic function, the normalized topological node similarity in (2.30) is used.

$$\tilde{s}(v_i, v_j) = \frac{s(v_i, v_j)}{\sum_{v_r \in G_k(v_i)} s(v_i, v_r)}. \tag{2.30}$$

With $\tilde{s}(v_i, v_j)$, we can define the characteristic function as follows:

$$\phi_{v_i}^{(k)}(t) = \sum_{v_j \in G_k} \tilde{s}(v_i, v_j)\left(\cos(ta_j) + i \sin(ta_j)\right) \tag{2.31}$$

Since this characteristic function is represented only at the node level, we can aggregate the characteristic functions over all nodes and obtain the following:

$$\phi_G^{(k)}(t) = \frac{1}{|V|} \sum_{v_i \in G} \phi_{v_i}^{(k)}(t) \tag{2.32}$$

Subsequently, we can sample $\phi_G^{(k)}$ at d evenly spaced points $t_1, \dots, t_d$ and merge them to the k-hop embedding:

$$\chi_{G_k} = \left[Re\left(\phi_G^{(k)}(t_i)\right), Im\left(\phi_G^{(k)}(t_i)\right)\right]_{t_1, \dots, t_d} \tag{2.33}$$

By aggregating all k-hop embeddings, we can compute the graph level embedding based on topological similarity as formulated in (2.34).

$$\chi_G = [\chi_{G_1}, \chi_{G_2}, \dots, \chi_{G_{k_{max}}}] \tag{2.34}$$

However, the final embedding X is constructed by concatenating the embeddings with transition probability using normalized topological similarity and the embedding with transition probability using normalized node influence. The above process can be repeated to obtain the embeddings with transition probability using normalized node influence.

2.3 Clustering

Clustering is a task of grouping a set of data points based on their similarity such that data points of the same group (called a cluster) are more similar to each other than data points from other groups (clusters). For the given task, the k-means algorithm was chosen. The k-means algorithm belongs to the partitioning category, where the dataset is separated into a specified number of clusters based on the similarity or distance among the data samples.

In the following subchapter, the mentioned algorithm is explained in more detail and performance evaluation metrics are presented to compare the results of the graph embedding techniques.

2.3.1 K-means

K-means is one of the most frequently used algorithms for unsupervised learning, especially in data science and statistics. The algorithm clusters data by separating samples in n groups of equal variances, minimizing the inertia or within-cluster sum-of-squares in (2.35). The number of clusters needs to be specified. The algorithm splits a set of N samples X into K disjoint clusters C, where each cluster is described by the mean μ_j of the samples. The means are also called "centroids" and they are not in set X. The problem with inertia is that it is not a normalized metric. It is only known that small values are good and zero is optimal, but in very high-dimensional spaces the Euclidean distances can become very large, so it is recommended to perform a dimensionality reduction algorithm prior to k-means clustering. [41]

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(\left\| x_i - \mu_j \right\|^2 \right) \tag{2.35}$$

The algorithm works as follows [42]:

1. Randomly select k samples from the dataset X as initial centroids
2. Assign each data point to the nearest cluster center
3. Compute new cluster centers as the average of all data points assigned to the cluster

4. Go to step 2 unless no improvement

To illustrate the procedure, two iterations of the k-means algorithm are shown in Figure 21 using an example. In the example, a set of 15 data points is examined and $k = 2$ clusters are expected to be identified. The left plot, two initial centroids were chosen, and each data point was assigned to the nearest cluster center. The plot in the middle shows how the centroids were recalculated by averaging all data points assigned to each cluster. As the new cluster centers better represent the middle of the clusters, the algorithm steps back to step 2 and assigns each data point to the closest recalculated centroid, which can be seen in the right plot of Figure 21. [42]

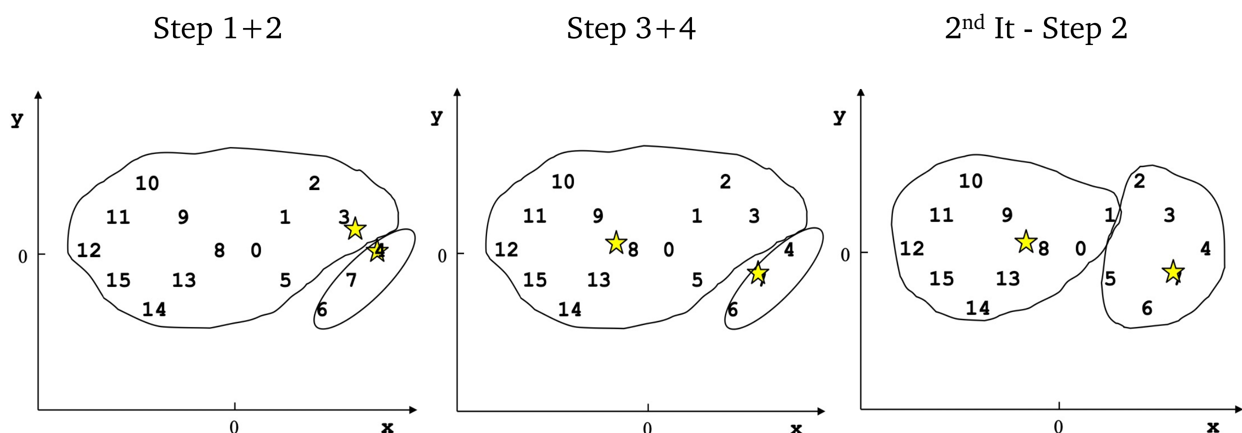


Figure 21. Example of clustering procedure with k-means algorithm [42]

To decide whether the k-means algorithm is suitable for a given task, we should take a closer look at the following advantages and disadvantages of this technique as pointed out by [42]:

Advantages:

- Simple and easy implementation
- Time and space complexity is $O(n)$
- Can scale up to large datasets

Disadvantages:

- Manually specify number of clusters k
- Due to random selection of initial centroids the results can vary
- Outliers cannot be identified, which can influence the clustering process where for instance an outlier can become a cluster itself

After analyzing the advantages and disadvantages of the k-means algorithm, it can be concluded that for the given task, which involves a large dataset, low time and space complexity is very important to complete the task in a reasonable time. The manual specification of the number of clusters is not a real disadvantage for the task, since the ground-truth data is already available and thus the number of clusters k is known. The simplicity of the algorithm is another advantage of k-means, which led to other clustering algorithms being disregarded in this work.

2.3.2 Performance Evaluation

To compare the clustering results between the different whole graph embedding techniques in 2.2, a performance measure is needed. The following four performance evaluation metrics are specifically designed for clustering algorithms. The first two metrics require the knowledge of the ground-truth labels, while the other two calculate a performance score based only on the graph embedding itself.

2.3.2.1 Rand Index (RI)

The RI is a measure of the similarity between two different data clusterings and is often referred to as accuracy. The RI does not ensure that a performance value close to 0 can be interpreted as random labeling. While the ARI corrects for chance and provides such a baseline. The RI is defined in (2.36), where the denominator is a binomial coefficient and C_2^n is the number of unordered pairs in a set with n elements. [41,43]

$$RI = \frac{a + b}{C_2^n} \tag{2.36}$$

For instance, in a set of 4 elements $\{A, B, C, D\}$, there are 6 unordered pairs: $\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}$. In this case C_2^n is equal to 6. If C describes the ground-truth class assignment and K the clustering, then a is defined as the number of pairs of elements that are in the same set in C as well as in the same set in K. While b describes the number of pairs of elements that are in different sets in C and in different sets in K. [41]

Given a set of 6 elements: $\{A, B, C, D, E, F\}$, the ground-truth data contains three clusters, where the first two elements correspond each to another cluster: {1, 1, 2, 2, 3, 3}. The clustering forms two clusters, the first three objects are in cluster 1 and the last three objects are in cluster 2: $\{1, 1, 1, 2, 2, 2\}$.

To calculate the RI, we need a, b, and C_2^n. In a set of 6 elements, there are 15 unordered pairs, which means $C_2^n = 15$. For a we need to find out how many pairs of elements are grouped together by both the ground-truth and the clustering. $\{A, B\}$ and $\{E, F\}$ are clustered together, so $a = 2$. b is

the number of pairs of elements that are not clustered together by both datasets: $\{A, D\}$, $\{A, E\}$, $\{A, F\}$, $\{B, D\}$, $\{B, E\}$, $\{B, F\}$, $\{C, E\}$, $\{C, F\}$, so $b = 8$. In total, the RI is equal to $\frac{2+8}{15} = 0.67$.

The RI can also be viewed as binary classification accuracy, where a is the number of pairs correctly labeled as belonging to the same subset (True Positives), and b the number of pairs correctly labeled as belonging to different subsets (True Negatives). The problem for clustering with multiple clusters is that the True Negatives can achieve a high number when both the predicted clusters and the ground truth data assign a pair different labels. In the example above that would mean that the pair $\{A, E\}$ of b only exists because the ground truth assigned them $\{1, 3\}$ and the prediction $\{1, 2\}$. For an increasing number of clusters it is fairly easy to achieve a high b, since the probability of assigning the the pair a different label is high. [44]

To solve the previously outlined problem, the ARI was invented (see (2.37)). The ARI utilizes the RI to calculate the similarity between two different clusterings, corrected for chance. When the ARI is equal to 1, the two clusterings are completely identical, and a value close to 0 shows a random label assignment. While the RI lies only between 0 and 1, the ARI may yield also negative values if the index is less than the expected value. [41,44]

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]} \tag{2.37}$$

2.3.2.2 Mutual Information (MI)

The MI between two random variables measures the non-linear relations between them. It represents the amount of information that can be obtained from one random variable by observing another random variable. The concept of mutual information is closely linked to entropy, as MI is also known as the reduction of uncertainty of a random variable if another is known. Therefore, a high MI value demonstrates a large reduction of uncertainty, while a low value demonstrates a small reduction. A MI value close to zero indicates that the two random variables are independent. [41,45]

The MI for two discrete random variables is defined in (2.38), where $P(i)$ and $P'(j)$ are the probability that an object picked at random from U falls into class U_i and equivalently for an object picked from V falls into class V_i. The probability $P(i, j)$ describes the probability that a randomly picked object falls into both classes U_i and V_i. [41,45]

$$MI(U, V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} P(i, j) \log\left(\frac{P(i, j)}{P(i)P'(j)}\right), \tag{2.38}$$

As previously mentioned, the MI is based on entropy. Entropy measures the level of expected uncertainty in a random variable. This is the reason why $H(U)$ describes how much information about the random variable U can be learned by observing a single sample (see (2.39)). Similarly, $H(V)$ shows how much information can be learned from the random variable V (see (2.40)). [45,46]

$$H(U) = -\sum_{i=1}^{|U|} P(i) \log (P(i)) \tag{2.39}$$

$$H(V) = -\sum_{j=1}^{|V|} P'(j) \log (P'(j)) \tag{2.40}$$

Like the RI, the baseline value of MI between two random clusterings tends to be larger when the partitions have a larger number of clusters. Therefore, the adjusted mutual information (AMI) can be calculated in a similar form as the ARI, analogous to 2.3.2.1 in (2.41). [47]

$$AMI = \frac{MI - E[MI]}{mean(H(U), H(V)) - E[MI]} \tag{2.41}$$

The expected value of the MI can be calculated using the equation in [47]. The AMI corrects the agreement between two random variables for chance. The AMI lies between 0 and 1, taking the value 1 when two partitions are identical and the value 0 when the MI between two partitions is equal to the value expected due to chance only. [41,47]

2.3.2.3 Silhouette Coefficient (SC)

When the ground-truth labels are unknown, the evaluation must be performed by the model itself. The SC describes such an evaluation that calculates whether points are clustered and separated well. The SC is defined in (2.42), where b represents the mean distance between a sample and all other points in the nearest cluster. While a stands for the mean distance between a sample and all other points in the same cluster. The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering. When the clusters are close to each other, the SC is close to zero. The score is higher if the clusters are dense and well separated, which is the standard concept of a cluster. [41]

$$s = \frac{b - a}{\max(a, b)}, \quad (2.42)$$

2.3.2.4 Davies Bouldin index

Similar to the SC, the Davies Bouldin index evaluates the clustering performance by the model itself. The Davies Bouldin index is defined as the average similarity measure of each cluster to its most similar cluster, where similarity is the ratio of intra-cluster distances to inter-cluster distances. Zero is the lowest possible value indicating better partition. Values closer to 0 indicate that the clusters are farther apart and less dispersed. The index has no upper bound, so it can reach infinity if the intra-cluster distances are very high.

The index describes the average similarity between each cluster C_i for $i = 1, \dots, k$ and its most similar one C_j. The non-negative similarity measure R_{ij} that is used for the Davies Bouldin index is defined in (2.43), where s_i is the average distance between each point of cluster i and the centroid of that cluster which is also known as the cluster diameter. d_{ij} is also the distance between cluster centroids i and j. After applying the similarity measure R_{ij} for every cluster C_i and taking the average, the Davies Bouldin Index is calculated as shown in (2.44). [41]

$$R_{ij} = \frac{s_i + s_j}{d_{ij}}, \quad (2.43)$$

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij} \quad (2.44)$$

3 Dataset

This chapter describes the dataset used to evaluate the whole graph embeddings. It starts with an overview of the dataset including information about the application domain, the objective, and the kind of data that is recorded. Subsequently, an Exploratory Data Analysis (EDA) was conducted (see Chapter 3.2) to understand the dataset in-depth, which is one of the most important steps before starting a ML project. This chapter corresponds to the first step of the CRISP-ML diagram in Figure 1, which deals with Business & Data Understanding.

3.1 Overview Bosch Dataset

This section presents an overview of Bosch's production line dataset by providing basic information about the dataset, the application domain, and the objective Bosch intends to achieve by releasing this dataset.

3.1.1 Application Domain

The dataset used in this scientific work is open-source and is originally derived from Bosch's production line. Bosch is a German company and a leading supplier of technology and services. Their mission is to develop products that are "Invented for life". To ensure their products' high quality and safety standards, they monitor the parts during the manufacturing process. More precisely, they record every single step of the assembly line during the manufacturing process of every single product. Since the progress of each part through the assembly line is recorded, Bosch is in an appropriate position to apply advanced analytics to minimize defective products in manufacturing. [48]

In August 2016, Bosch published a dataset on the online community platform Kaggle and launched a competition to predict defective products using thousands of measurements along the assembly line [49].

3.1.2 Dataset Overview

For the purpose of using ML models, the dataset is already distributed into a training and test dataset. The dataset has a size of 14.3 GB and records the progress of exactly 1,184,687 products during the manufacturing process. The learned model will be used to predict whether a product is defective or not based on a dataset of 1,183,748 products. The data is categorized into three main feature types, namely: categorical, numerical, and date features.

The names of all features in the dataset follow a certain pattern. Figure 22 shows that each numerical feature name contains information about the production line, the station, and the corresponding feature number. In comparison to date features the nomenclature is similar except for the last value that contains the letter "D" instead of "F". The "D" stands for date and represents a date value.

Each of the 1,184,687 products has its unique product id and its corresponding features. Many features contain “NaN” values. When the product passes a station, the feature stores a value (numerical, categorical, or date), otherwise the feature displays “NaN”.

In this work, the goal is to evaluate graph embedding techniques on Bosch’s dataset. For the use of ML algorithms, it is not only necessary to understand the meaning of the data points, but it should also be possible to develop an expressive KG capable of producing a solid performance for the clustering task. Due to a lack of understanding of the categorical features, they are neglected in this documentation as well as in the software implementation.

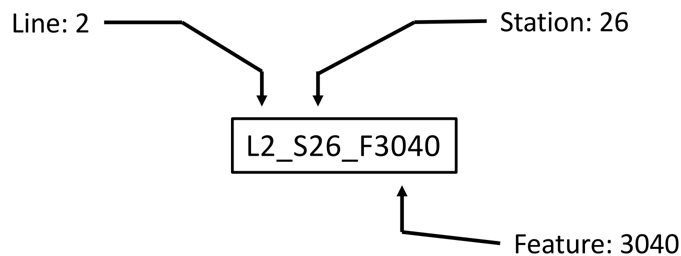


Figure 22. Nomenclature for numerical features

Production lines in this context mean that the input materials undergo a series of manufacturing steps, approaching the finished product from step to step. Within each production line, there are stations where various operations are executed and values are also measured. After examining the dataset, we can draw the conclusion that the production line incorporates 4 lines and 52 stations. In Table 1 each line is represented with its corresponding station. It can be observed that the production lines are separated from each other. This means that each station can belong to only one production line.

Table 1. Assignment of stations to their corresponding line

Line	Stations
L0	0 - 23
L1	24 - 25
L2	26 - 28
L3	29 - 51

3.1.3 Bosch's Objective

Bosch's dataset is incredibly large due to the various production lines, product diversity, and the number of products. Therefore, it is almost impossible for a company to track internal product failures manually. When working manually, it is very time-consuming and costly to ensure good product quality. Consequently, Bosch aims to develop a ML algorithm that predicts internal failures to improve the company's product quality at a lower cost. [49]

This work does not go so far as to make a binary classification of whether the product fails or not. In this work products are clustered into different product variants and afterwards compared with their ground-truth data. The failure prediction is beyond the scope of this work. It is more important to develop a ML model that achieves solid performance by using graph embeddings. The reason why performance is so important is that the cost of misclassification to an organization can be very high. If a model predicts that a product will have no internal failures in the future, then the prediction must be very accurate.

3.2 Exploratory Data Analysis

The notion of EDA is based on an important principle. It is necessary to understand what can be done with the given data before measuring how well it is done [28]. This principle enables scientists and engineers to tackle data science projects more easily and effectively. EDA is designed to empower people to make useful discoveries from data, especially in the absence of prior assumptions or research. The main goal is to maximize insights into the dataset [50]:

- Reveal underlying structure
- Detect outliers and anomalies
- Test underlying assumptions
- Find suitable models

In this subchapter, the numerical and date features of the Bosch production line are examined in more detail. For simplicity, only the training dataset with over 1.1 million rows is considered in this chapter. The categorical features have been neglected because of a lack of explainability. The implementation of the EDA is inspired by [51].

The research group for Management of Industrial Production (MiP) of the Institute of Production Management, Technology and Machine Tools (PTW) at the Technical University of Darmstadt (TUDa) uses a workstation to handle large amounts of data. The workstation's CPU consists of an AMD Ryzen 9 3900X 3.8GHz 12-Core processor with 32GB RAM. The GPU contains two NVIDIA Titan RTX 24GB GDDR6. To load the data more efficiently, more than 1.1 million products are loaded in chunks.

3.2.1 Numerical Features

The dataset includes 968 numerical features, and the main challenge is to extract meaningful information from it. It contains a final column that indicates the result of the product quality check. The result is called the "Response". It holds a binary variable, where 0 means the product passed the quality check and 1 means it was rejected. First, it is checked whether the dataset is balanced or unbalanced by looking at the "Response" column of the numerical training set. The products are assigned to either the "OK" or "NOK" class (see Figure 23).

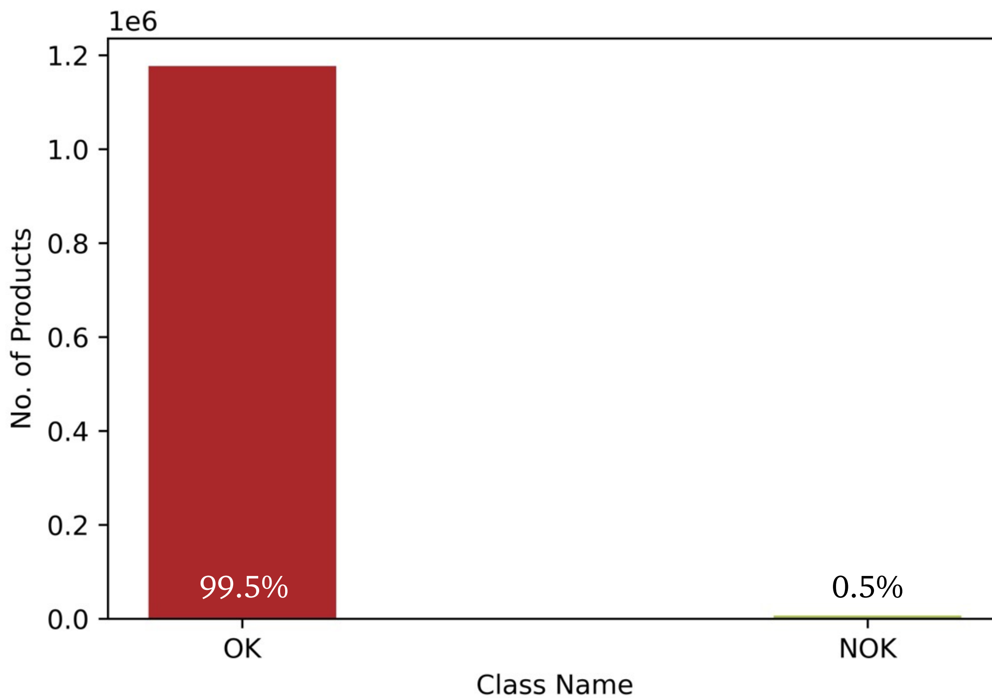


Figure 23. Number of products classified as functioning or failed

It can be noted that the dataset is highly unbalanced because nearly 99.5% of all assembled products do not have any internal failures.

To get a better understanding of the data, it is necessary to determine the number of measurements and products passed through each line and each station. The number of product failures and the error rate could also be useful resources. Figure 24 shows the number of measurements for each production line.

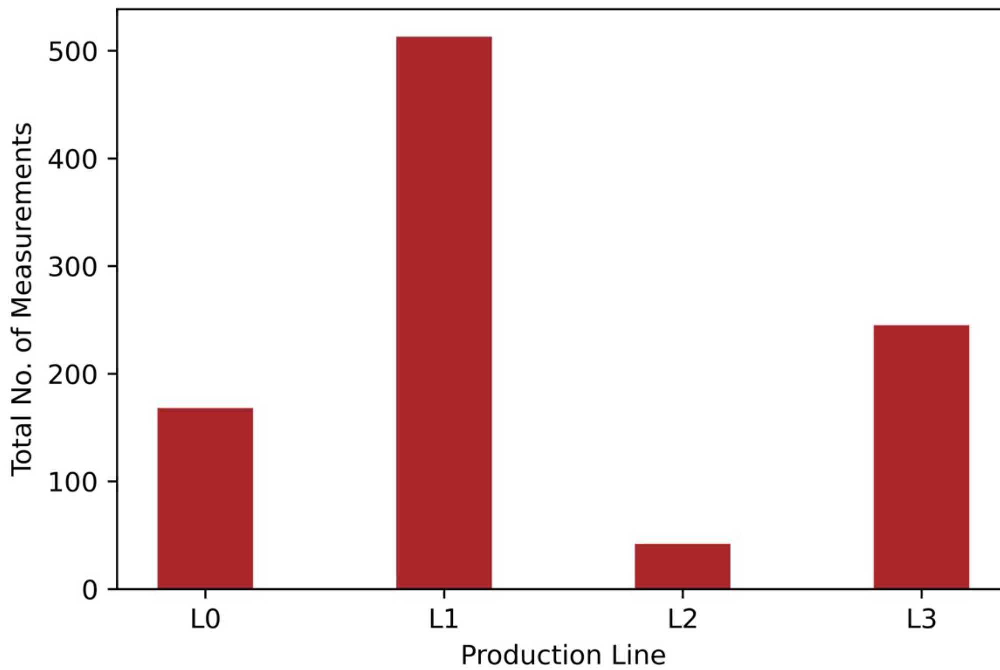


Figure 24. Number of measurements for each production line

The number of features per production line is summed up and summarized in Figure 24. Analyzing the figure makes it clear that L1 conducts the most measurements. As already known from Table 1, L1 has only stations 24 and 25, so it can be assumed that both stations are very large. These stations may also be very demanding as they make more than 500 measurements in total. In contrast, L2 behaves completely differently, as it also contains only 3 stations, but performs less than 50 measurements, which may mean that these stations are simple in design.

However, this illustration only provides information about the number of measurements in each production line, while Figure 25 shows the absolute number of products passed through each station. It is essential to remark that the numerical dataset does not contain information about S42 and S46, which is the reason why the scale has jumped from S40 to S47. Although L1 runs numerous measurements, not many products pass through this line. It can be concluded that L1 has the highest number of measurements per product. Also, L0 and L3 are the lines with the highest number of products passed through.

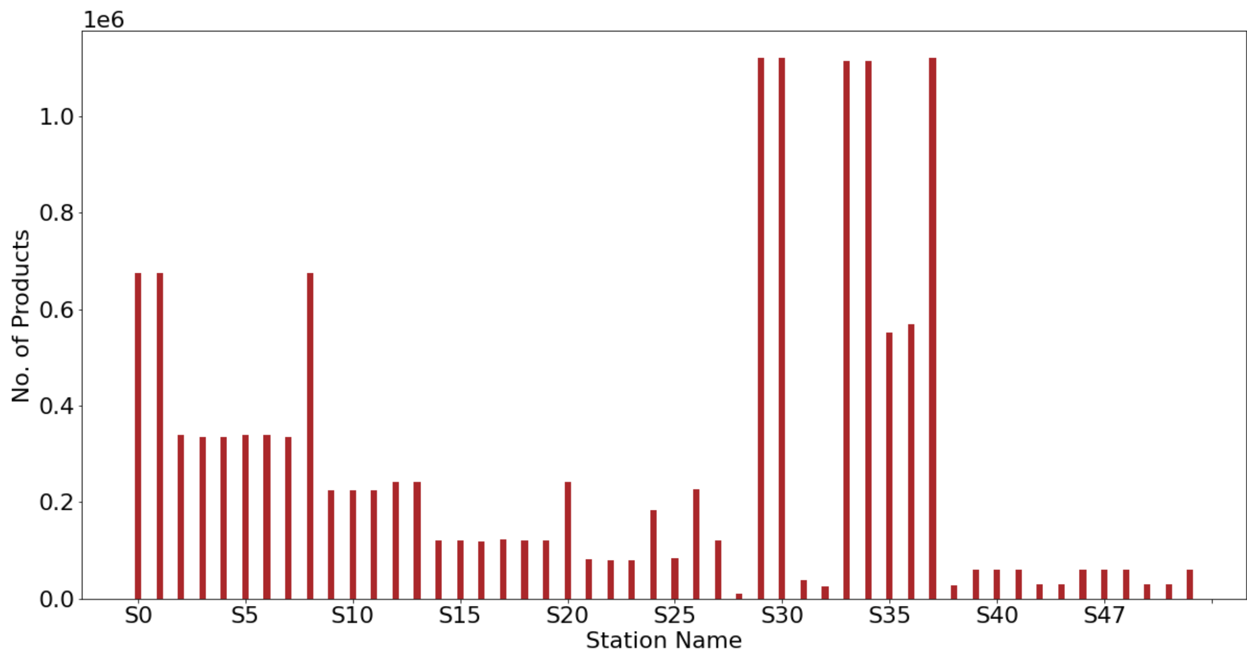


Figure 25. Number of products passed through each station

After analyzing how many products visited each station, it can be taken one step further by examining the failed products of this manufacturing process. Figure 26 shows the number of product failures for each station. The data in this figure can be compared to Figure 25, which shows a very similar trend. The number of products passed through S29 until S37 is significantly high compared to the other stations, while the number of defective products for the same stations is also very high in comparison. In summary, both figures demonstrate that the larger the number of products, the higher the number of product defects.

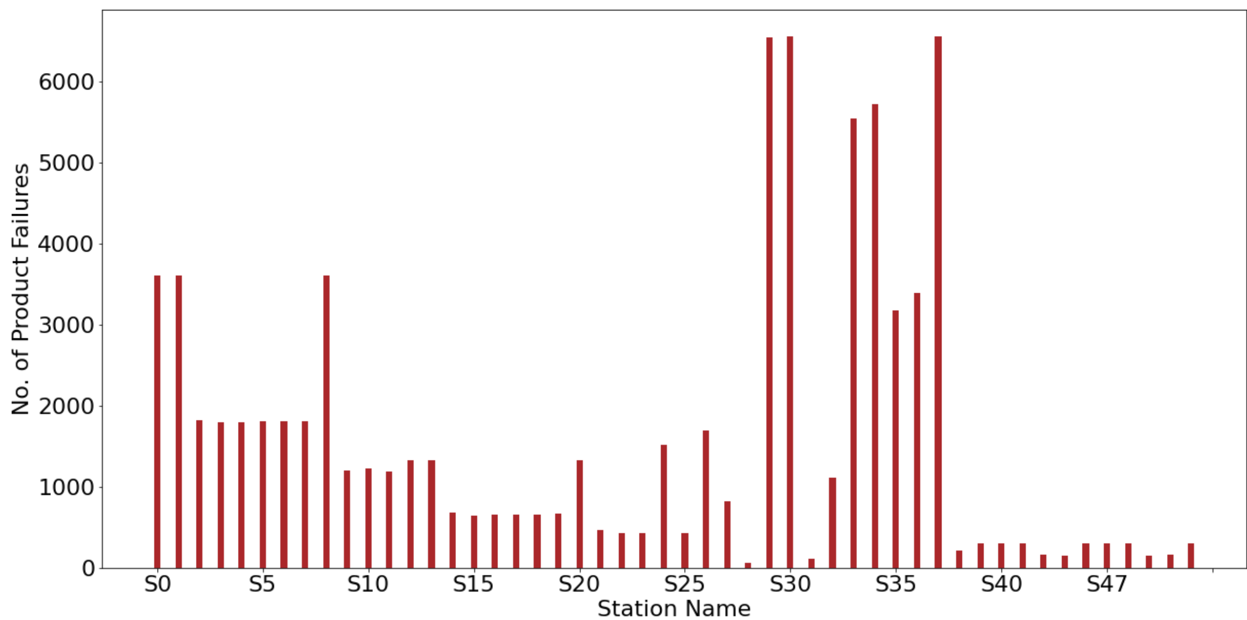


Figure 26. Number of product failures for each station

We cannot declare a station to be defective based only on the absolute number of product failures per station. Since each station works with a different number of products, it is important to look at the error rate for each station. Figure 27 shows the error rate as a percentage for each station. The figure shows that the error rate for all stations is less than 5%, with all stations except S32 having a similar error rate. Even though S32's error rate is about eight times higher than the other stations, it should be noted that S32 processed just under 30000 products, while S29, for instance, processed more than 1 million products. Assume the prediction of internal failures will be made with the raw data, then the values of S32 should be used as one of the key features for predicting the internal failures of a product.

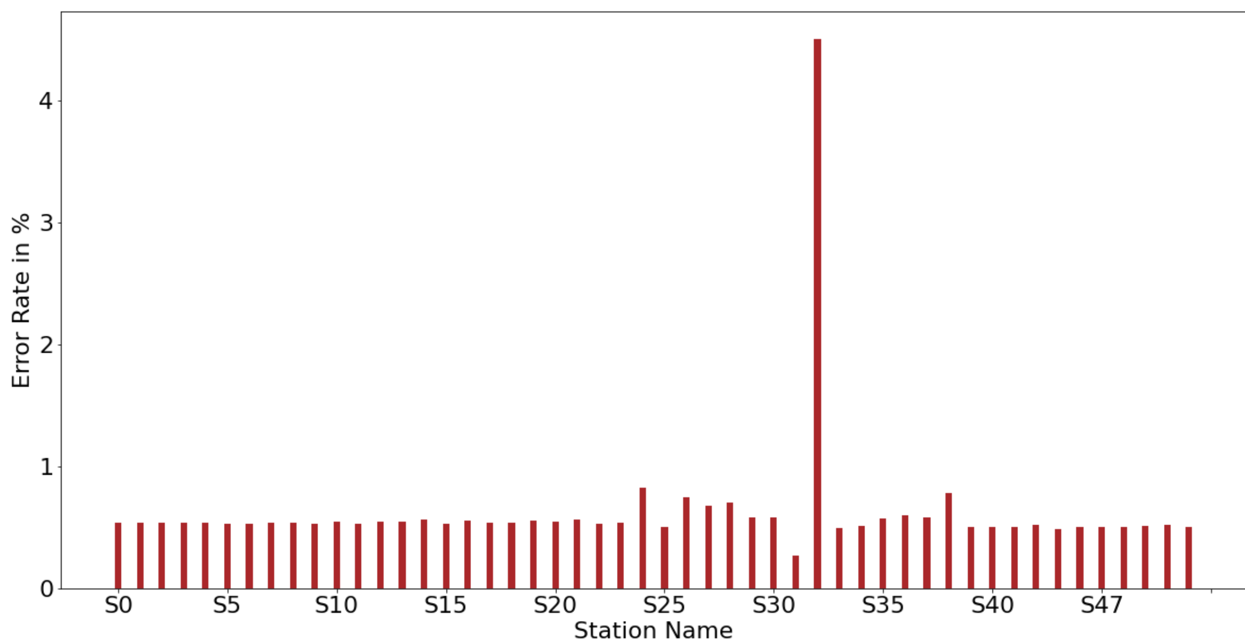


Figure 27. Error Rate for each station

Another noteworthy aspect is the high number of non-zero measurements along with the numerical dataset. Figure 28 shows the percentage of NaN values for each feature. The figure reveals that most features contain more than 90% NaN values, this may be an issue when performing the classification task. Therefore, within the scope of this work, a clustering analysis will be carried out first and classification will be aimed at in further investigations. However, there are fewer features of L2 and L3, where the percentage of NaN values is only about 5%.

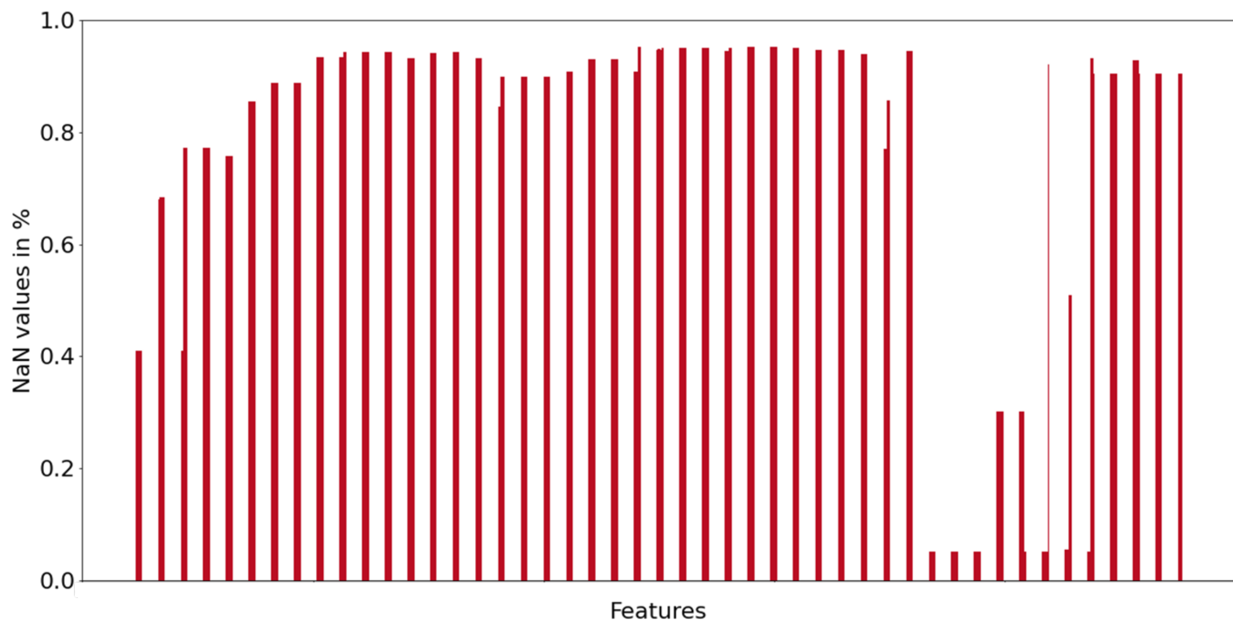


Figure 28. The proportion of NaN values vs. Feature

3.2.2 Date Features

The date dataset contains 1156 date features, with many missing values. The analysis of this data is crucial to understand production time and product complexity, which are useful resources for identifying internal failures. Skimming the data, one may notice that the same stations often have the same date value. The date feature has similarities to time series data, which is periodic in nature. To find a periodic pattern in this data, Figure 29 illustrates the number of all date records across the values of the date feature.

It can be concluded that the date values range from 0 to 1718 with a granularity of 0.01. To understand what time span is addressed by the date value range, a so-called autocorrelation function can be calculated. The autocorrelation describes the relationship of a time series to its previous versions in time. It computes the correlation with a k'th lagged version of itself. In this use case, the granularity of 0.01 is chosen for k. Figure 30 shows the autocorrelation with lag ($k = 0.01$) over the values of the date features [52,53].

It can be perceived that the largest peaks, shown in red, are at about 16.75 ticks, which corresponds to one week. Within a period, there are seven local maxima at 2.39 each, corresponding to seven days per week. Since a week is equivalent to 16.75 ticks, the granularity of 0.01 ticks is equivalent to 6 minutes. Thus, the data is recorded at a granularity of 6 minutes. To sum up, the data was recorded over a period of 102.6 weeks (1718 ticks), which is the reason for the variability within the dataset [52].

PTW
TU DARMSTADT

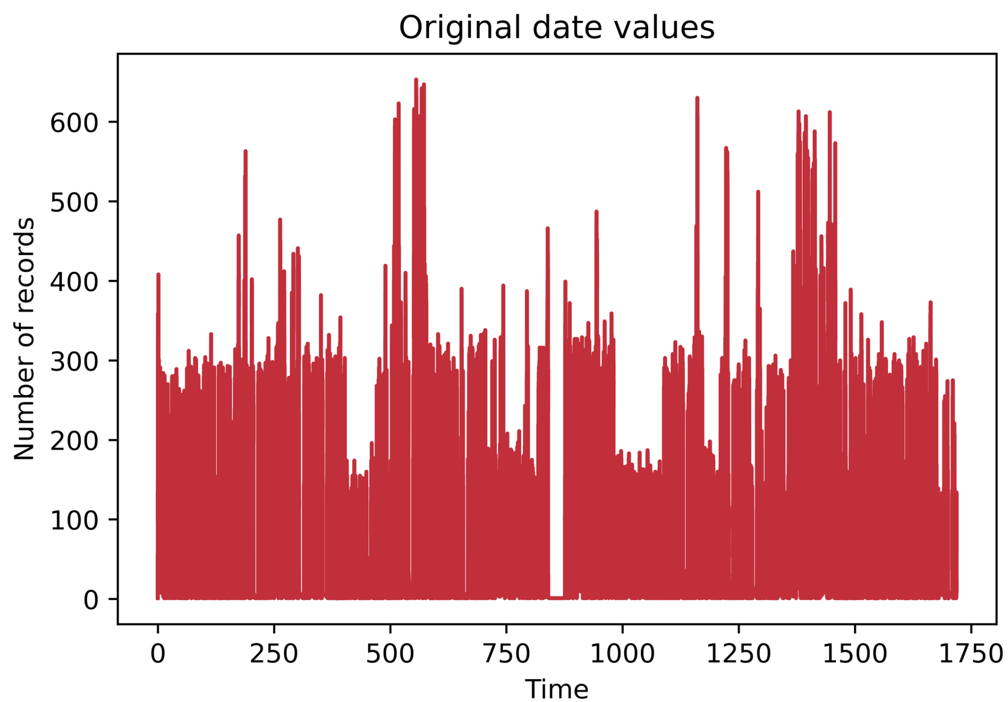


Figure 29. Number of records for date features over time

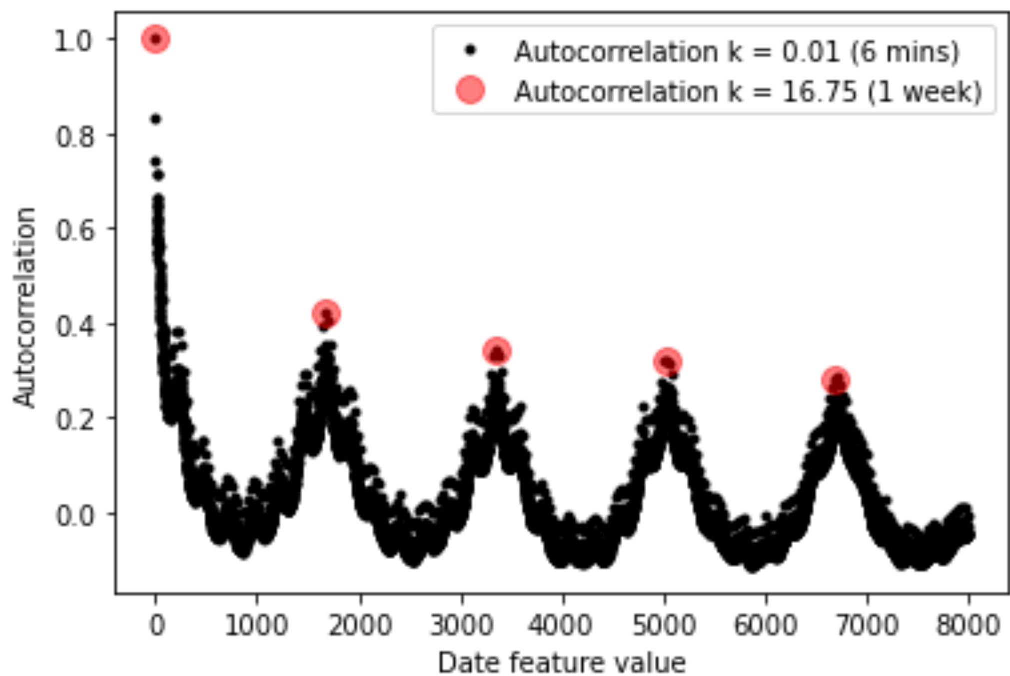


Figure 30. Sensor Time Autocorrelation

Since the data was recorded during the real manufacturing process, it can be useful to take a closer look at the time required for a product. For manufacturing we can assume that the more time a product takes, the more complex it is. Therefore, further investigations were made to see whether

the time difference has an impact on the subsequent failure of the product. Table 2 shows the time difference for different percentiles and product classes.

Table 2. Total time spent on a product for different product classes

Percentile	Class 0	Class 1
25th	1.71	1.95
50th	3.69	4.93
75th	11.8	16.3
90th	35.1	42.6
99th	62.9	72.4

The figure reveals that for each percentile, product class 1 has a higher total time value. Thus, the more time it takes to manufacture a product, the higher its product complexity and therefore the more likely it is to contain internal failures. Figure 31 illustrates the results as a violin diagram with a logarithmic y-scale. [51]

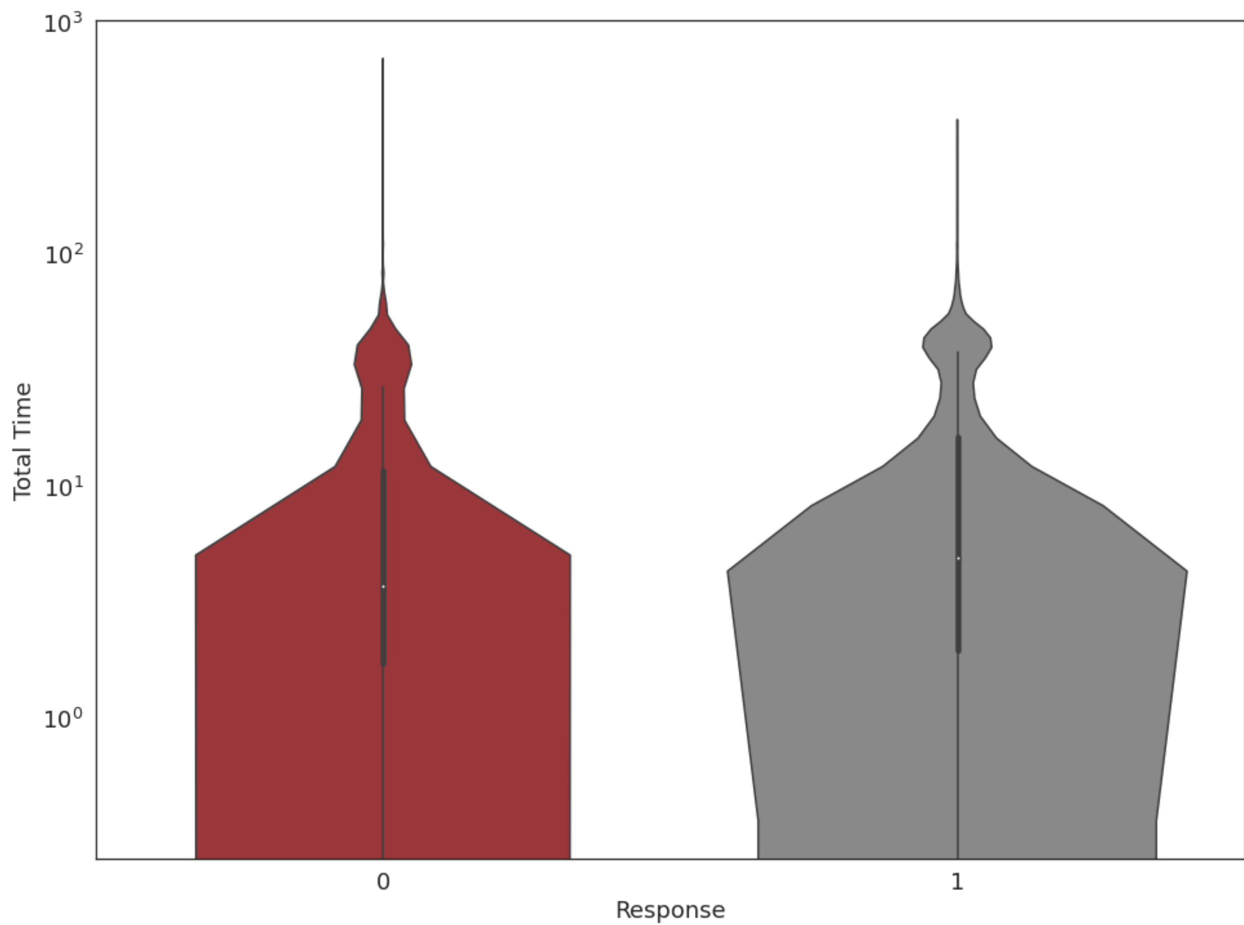


Figure 31. Violin Plot on total time spent on a product for different product classes

All in all, the following important findings can be drawn from the EDA [51]:

- Unbalanced dataset with more than 99.5% products that are not defective
- Significant number of NaN values are present for both numerical and date features
- Station S32 shows the highest error rate
- Date feature follows periodic pattern
- Date values range from 0 to 1718, which corresponds to approximately two years
- Positive correlation between time difference and error rate

4 Experiment

Based on the information about whole graph embeddings in Chapter 2 and the findings from the use case dataset in Chapter 3, this chapter explains the implementation of the experiment for the clustering task using whole graph embeddings as input features. This chapter starts with the preparation of the dataset and then explains the workflow to conduct the clustering analysis. Subsequently, the used graph model is presented, and it is elaborated on how the graph embedding techniques are applied to the KG created. Finally, the implementation of the clustering algorithm is illustrated in more detail.

4.1 Preparation

This section provides an overview of the technical environment used for this experiment. Furthermore, the process of the raw data preparation is explained, which demonstrates the second step of the CRISP-ML diagram in Figure 1.

4.1.1 Technical Environment

The experiment in this chapter is executed by a server of PTW. More precisely, it was the workstation of the research group MiP. It is the same workstation that was used for the EDA in Chapter 3.

The remote connection was established via an SSH tunnel in the open-source Visual Studio Code Integrated Development Environment (IDE). Visual Studio Code was chosen as IDE because it natively supports both IPython Notebooks (.ipynb) and simple Python files (.py). The advantage of the mentioned notebooks is that they can contain both code and rich text elements, such as figures and equations. It is also much more convenient to have multiple figures on the same page, compare them, and move the cells.

The software implementation of this work is based on three different tasks (see Figure 32). Namely, data exploration, data preprocessing, and the implementation of graph embedding techniques followed by clustering. The data exploration task is already presented in Chapter 3. The data preprocessing task is outlined below.

```
master_thesis
├── data_exploration
|   ├── exploratory_data_analysis.ipynb
|   └── plots
├── data_preprocessing
|   ├── unique_paths_to_final_df.ipynb
|   ├── final_df_to_graph_dict.ipynb
|   └── pkl_files
|       ├── feature_df.pkl
|       ├── directed_graphs.pkl
|       └── undirected_graphs.pkl
|       └── ground_truth_data.pkl
├── graph_embeddings
|   ├── main.py
|   ├── embedding_original_techniques.py
|   ├── embedding_modified_techniques.py
|   ├── clustering.py
|   ├── bosch_data
|   └── embedding_techniques
└── baseline_clustering
    └── raw_data_clustering.ipynb
```

Figure 32. File Structure of the software implementation

4.1.2 Data Preprocessing

After examining the Bosch dataset, it was determined that there are thousands of different products. To simplify the evaluation of graph embedding techniques, we continue to use only the numerical dataset. The approach of applying a clustering model to the generated graph embeddings requires a number of clusters as input.

Therefore, a student researcher of the PTW worked to determine the number of unique paths for the numerical dataset. The person found out that there are 16,710 unique paths, which is equivalent to a production of 16,710 different products. Each product of the same product variant was then stored in a unique product path. The data was stored in so-called pickle (.pkl) files. Pickle files are used to convert Python object structures into binary files on disk.

The task was formulated to take a certain number of product variants and use them to create graphs for each product inside of these variants. Figure 33 shows the workflow for data preprocessing. First, 4 out of 16,710 product variants were chosen that have a small number of common features. The selection was based on the number of common features between pairs of all four product variants and the number of features per variant. Thus, the most diverse product variants were chosen. After determining the relevant products, the second step was to merge the numerical data of all these products into one large numerical dataset.

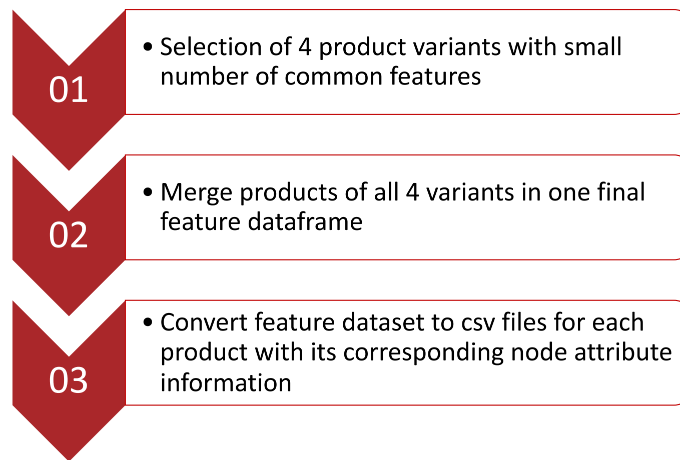


Figure 33. Workflow Data Preprocessing

In the numerical dataset, each row holds data for only one product (see Table 3). To decouple the data of each product from the others, CSV files were created for each product. The numeric CSV files contain information about the measured feature value and the corresponding line, station, and feature Id (see Table 4). Each product includes about 150 – 250 feature values, which means that each CSV file contains this number of lines. These CSV files are further used to facilitate the creation of graphs.

Table 3. Excerpt from numerical training set

Index	Id	L0_S0_F0	L0_S0_F2	L0_S0_F4	L0_S0_F6
0	4	0.03	-0.034	-0.197	-0.179
1	6	NaN	NaN	NaN	NaN
2	7	0.088	0.086	0.003	-0.052
3	9	-0.036	-0.064	0.294	0.33
4	11	-0.055	-0.086	0.294	0.33

Table 4. Excerpt from the numerical CSV file of product Id 4

Index	lineId	stationId	featureId	featureValue
0	L0	L0_S0	L0_S0_F0	0.03
1	L0	L0_S0	L0_S0_F2	-0.034
2	L0	L0_S0	L0_S0_F4	-0.197
3	L0	L0_S0	L0_S0_F6	-0.179
4	L0	L0_S0	L0_S0_F8	0.118

4.2 Workflow

This subsection explains the next steps of the experiment which corresponds to the modeling step of the CRISP-ML diagram in Figure 1. The workflow is shown in the flowchart in Figure 34. The result of the data preprocessing was 14,353 CSV files for the feature values. These files were used to create a graph for each product, i.e., 14,353 graphs were created. For the graph creation, two main approaches were analyzed.

The first was to create graphs using the No-SQL graph database Neo4j. The platform is very user-friendly and great for graph visualization. This approach required working with the client-library Py2neo. The graph definition was done in Python using the Py2neo library and the connection between Python and Neo4j can be established via Bolt or HTTP. Once a graph is created, it can be converted into a GraphML (.graphml) file, which can be easily processed using the Python package NetworkX. The graph is saved as a NetworkX graph, which is a required input format for graph embedding techniques. The problem with Neo4j is that each database can only have one graph. Having multiple databases in Neo4j is only possible with an enterprise license, which is not given in this scientific work. A possible solution could be to create different subgraphs within one database, but the implementation has to be done in Neo4j's graph query language *Cypher*. This shows that there are many obstacles when working with a graph database, leading to the conclusion to follow the second approach and perform the graph definition entirely in NetworkX.

As already mentioned, the second approach was pursued. When a graph is created, it can be specified whether it should be a directed or undirected graph. In this work, the goal is to evaluate the graph embedding techniques considering directed and undirected graphs. However, Karate Club's graph embedding techniques are specifically designed for undirected graphs. Although they are designed for undirected graphs, Feather-Graph, Wavelet Characteristic, and Geo Scattering allow to create attributed graphs, therefore, these libraries are modified by adding node attributes and can after the modification be used for the creation of either directed or undirected graphs. Since the Laplacian matrix is different for directed and undirected graphs, we need to adjust the libraries for NetLSD, FGSD, SF, and IGE in order to create directed graphs. Graph2Vec, GL2Vec, and LDP can be applied to both directed and undirected graphs without any adjustment.

For the generation of graph embeddings, a list of all 14,353 graphs is passed to the graph embedding techniques, which outputs an embedding vector for each graph. The size of the embedding vector varies depending on the technique, and the default size can also be adjusted for some techniques.

Subsequently, the graph embeddings are used to train a clustering model. The model is initialized with four clusters, since only four product variants are considered in this work. The clustering is evaluated using four performance evaluation metrics, from which two require ground-truth data.

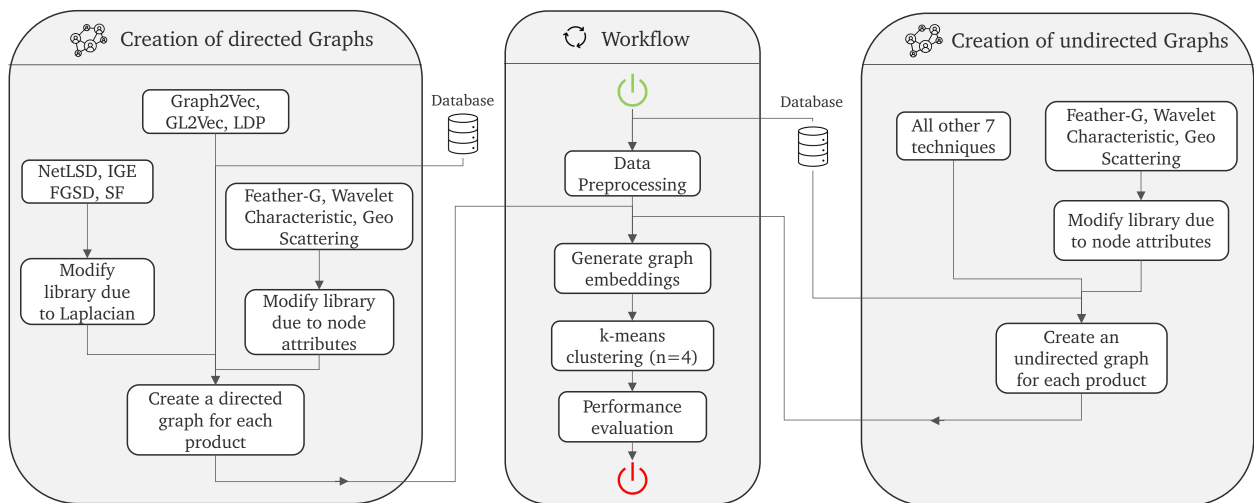


Figure 34. Flowchart of the workflow after data preprocessing

4.3 Graph Model

This section starts with introducing the KG data model, continues with the directed and undirected graph model, and finally explains the implementation of the graph embeddings.

4.3.1 Data Model

Graph representation plays a very important role in graph embedding design. If non-expressive graphs are designed and graph architectures that are not understandable are used, the cluster model will not be very promising and the graph embedding evaluation will not be beneficial.

For the purpose of this thesis, a KG is utilized to represent all the information about the manufactured products. This means that the graph should show the numerical features related to each station and line and their corresponding ID. Figure 35 shows the data model for the KG.

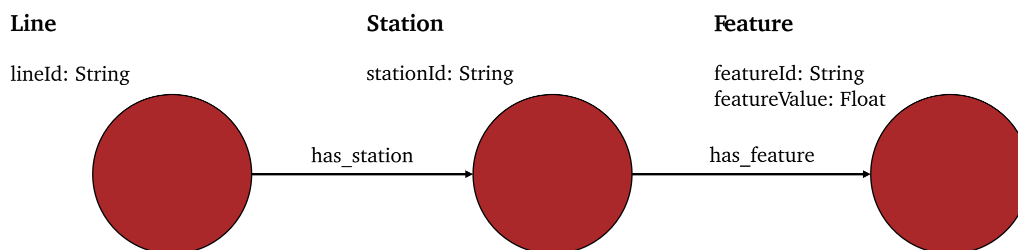


Figure 35. Data model for the Knowledge Graph [5]

The figure displays three different entities: Line, Station, and Feature. The lines are specified by the “lineId”, similar to the stations defined by the “stationId”. While features are identified by the identifier “featureId”, and the “featureValue” that stores the sensor record. [5]

The data model also contains two types of relationships, namely: “has_station” and “has_feature”. The has_station relationship demonstrates which stations belong to the line. Similarly, the has_feature relationship describes in which station the feature was measured. [5]

4.3.2 Directed Graph Model

The directed graph model shows an example implementation of the data model. The Python package NetworkX was used to implement the data model. The wide range of graph functions of NetworkX allows entity properties and relationships to be represented as node attributes and edges.

Figure 36 shows an example of a directed graph model. It can be perceived that edges lead from line nodes to station nodes and from station nodes to feature nodes.

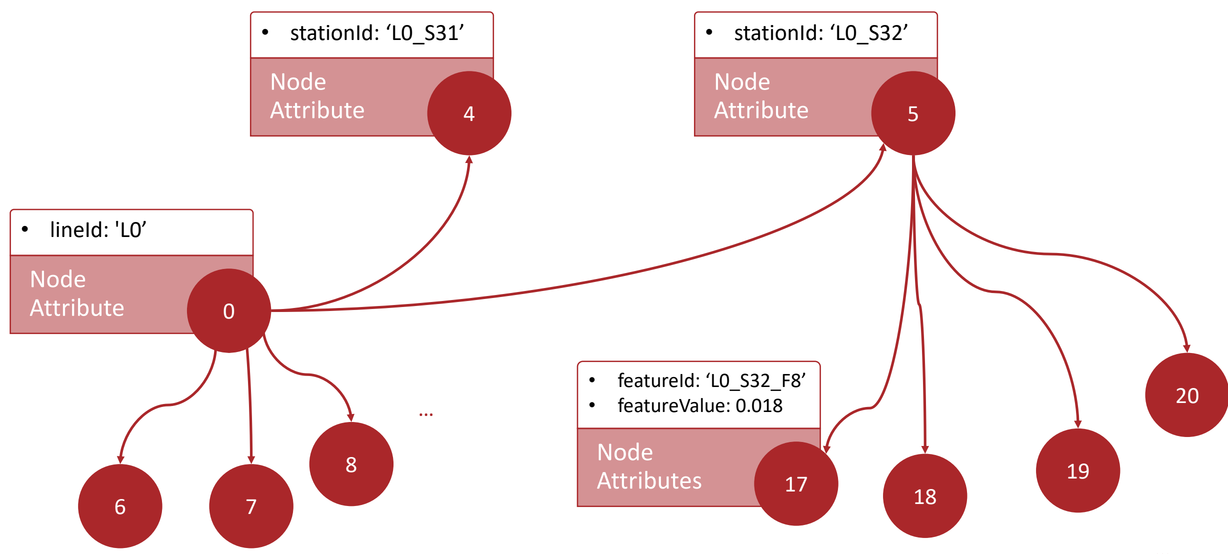


Figure 36. Example of the directed graph model

4.3.3 Undirected Graph Model

The undirected graph model depicts the same graph model as 4.3.2, with the exception that each edge is undirected. This means that the hierarchy of an edge leading from a line to a station cannot be maintained. Similarly, the hierarchy of a station resulting in a feature cannot be preserved. It is

also not possible to distinguish at which station the product started and after which station the product was completed. In the software implementation, the difference is in the creation of empty NetworkX graphs, working with the function networkx.Graph() instead of networkx.Digraph().

4.3.4 Implementation

The implementation of the directed and undirected graph models is very similar. Therefore, the following implementation is explained for both graph models.

As illustrated in Algorithm 1, a dictionary of graphs is to be built for each product. First, each product is looped, and empty graphs are initialized. Then, the number of nodes needed for each graph is calculated by taking the sum of all unique lineIds, stationIds, and featureIds. This number of nodes is used to add that number of nodes to a graph. It is important to note that the names of the nodes are integers only and count from 0. In line 7 the node attributes are defined using information from the CSV files.

For the declaration of each edge, a function is written to determine the dictionary pairs. The has_station function iterates through the line and station dictionaries and checks if the first letters of the stationId match the lineId. If a match is found, they are appended as a tuple to the has_station dictionary. A similar procedure is performed for the has_feature function, except that it checks whether the first letters of the featureId are the same as the stationId. Finally, the function transfer_to identifies the sequence of numeric node Ids that a product has passed through, and the edge is declared as a weighted edge with the transfer time as the weight. The output of the algorithm is the dictionary G.

ALGORITHM 1: GRAPH CREATION FROM CSV FILES (FNAME)

```
    input:   fname: Path to CSV feature files
             no_products: Number of manufactured products
    output:  G: Dictionary of all created graphs
1   begin
2   |   Initialization: empty lists and dictionaries for nodes, node attributes, edges, and edge
    |   attributes
3   |   for i = 0 to no_products do
4   |   |   create multiple empty graphs G[i]
5   |   |   compute no_nodes for each graph by taking the sum of the set of lineIds,
    |   |   stationIds, and featureIds
6   |   |   add to each graph its corresponding no_nodes
7   |   |   set node attributes with information about the lineId, stationId, featureId, and
    |   |   featureValue
8   |   |   FUNC: has_station that finds to each line its corresponding stations
9   |   |   FUNC: has_feature that finds to each station its corresponding features
10  |   |   add to each graph the edges has_station and has_feature
11  |   return G
```

Algorithm 1. Graph creation from CSV files

To get a better understanding of how the nodes of a single graph look like, Figure 37 shows an excerpt of the first 15 nodes with their corresponding node attributes. It can be observed that the first nodes of the graph belong to the entity line, the following ones are station nodes and finally all feature nodes are listed.

```
Node 0 : {'lineId': 'L3'}
Node 1 : {'lineId': 'L1'}
Node 2 : {'lineId': 'L2'}
Node 3 : {'stationId': 'L2_S26'}
Node 4 : {'stationId': 'L1_S24'}
Node 5 : {'stationId': 'L3_S30'}
Node 6 : {'stationId': 'L3_S29'}
Node 7 : {'stationId': 'L3_S35'}
Node 8 : {'stationId': 'L3_S34'}
Node 9 : {'stationId': 'L3_S37'}
Node 10 : {'stationId': 'L3_S33'}
Node 11 : {'featureId': 'L1_S24_F1512', 'featureValue': 0.065}
Node 12 : {'featureId': 'L1_S24_F1514', 'featureValue': -0.034}
Node 13 : {'featureId': 'L1_S24_F1516', 'featureValue': -0.105}
Node 14 : {'featureId': 'L1_S24_F1518', 'featureValue': -0.001}
```

Figure 37. Excerpt from the first 15 nodes of product id 4's graph

4.4 Implementation of Graph Embeddings

After introducing the whole graph embedding techniques in 2.2, the implementation of those techniques for the given task is described in this subchapter. Note that the Karate Club library proposes one possible implementation of the techniques in 2.2 which is suited to their benchmark datasets and has certain limitations. Based on the use case the techniques need to be modified which was conducted with most of the techniques in accordance with our graph model. The Karate Club library has the following limitations:

- Graphs are undirected and not multipartite, while the nodes are homogenous, and edges are unweighted
- Nodes are indexed with integers consecutively counting from 0
- Not all techniques consider node attributes

The graph model, defined in 4.3, consists of directed, not multipartite graphs with attributed nodes and unweighted edges. The nodes are also indexed with integers counting from 0. The problems we are facing because of Karate Club's limitations were addressed except for the limitation that our nodes should be homogenous. Since the nodes represent information about three different entities lines, stations, and features, the nodes are inhomogeneous. However, the two main issues regarding the graph's orientation and the node attributes were solved and are explained in the following.

4.4.1 Graph Orientation

The whole graph embedding techniques SF, FGSD, NetLSD, and IGE require the Laplacian matrix. Since we define the graph as directed, the standard built-in library of NetworkX cannot process asymmetric matrices as it is the case for undirected graphs. Therefore, we propose to calculate the Laplacian matrix as well as the normalized Laplacian matrix by the explained equations in 2.1.3. First, we have to distinguish between the four above mentioned techniques. SF, FGSD, and NetLSD rely on the normalized Laplacian matrix, while IGE needs only the regular directed Laplacian matrix. Surprisingly, the NetworkX library contains a function that calculates the regular Laplacian matrix for directed graphs. Accordingly, the function "_get_spectral_features" as well as the function "_get_histogram_features" of the class IGE were modified.

The other techniques depend on the normalized Laplacian matrix, which is for directed graphs distributed into out-degree left and in-degree right normalized Laplacian matrices (see Figure 11). Since the out-degree left normalized Laplacian matrix is more common and is related to the random walk matrix, this matrix is calculated for the mentioned techniques with Algorithm 2. The algorithm begins with the initialization of the "nodelist" (list of nodes), and proceeds with the calculation of the adjacency matrix in CSC format as explained in 2.1. Subsequently, the outdegree diagonal matrix is calculated and used for determining the out-degree left normalized Laplacian matrix.

ALGORITHM 2: CALCULATE DIRECTED NORMALIZED LAPLACIAN	
1	**begin**
2	nodelist = range(graph.number_of_nodes())
3	A = nx.to_scipy_sparse_matrix(graph, nodelist = nodelist, format = "csc")
4	n, m = A.shape
5	diags = A.sum(axis = 1) # axis =1 → outdegree matrix
6	D = sp.sparse.spdiags(diags.flatten(), [0], m, n, format = "csc")
7	I = np.identity(len(graph))
8	$\hat{L}$ = I − inv(D) · A
9	**return** $\hat{L}$

Algorithm 2. Calculate directed normalized Laplacian matrix

4.4.2 Node Attributes

The three newer graph embedding techniques GeoScattering (2019), Feather-Graph (2020), and Wavelet Characteristic (2021) are able to process a feature vector X. The Karate Club library already designed a feature vector X considering the node neighborhood and the clustering coefficient. In order to let the graph embedding understand the node attributes of the defined graph, we create a node attributes vector in Algorithm 3. The node attributes used for this array are extracted from the "featureValues" (feature values) and are thus completely numeric. Unfortunately, the given graph embeddings are not able to process node attributes that contain strings, which is why the lineIds, stationIds, and featureIds cannot be considered. However, the relation between those entities is reflected in the edges. In line 1, the algorithm stores the featureValue attribute of each node in an array. Recalling the graph structure, we observe that only the last nodes of a graph have a featureValue. Therefore, in line 3 the beginning values that normally correspond to lines and stations are filled with zeros.

ALGORITHM 3: CREATE NODE ATTRIBUTE ARRAY	
1	**begin**
2	attributes = np.fromiter(nx.get_node_attributes(graph, "featureValue").values())
3	n = graph.number_of_nodes() – len(attributes)
4	attributes =np.pad(attributes, (n, 0), "constant", constant_values = 0).reshape(-1,1)
5	**return** attributes

Algorithm 3. Creation of a node attribute array

The remaining three techniques Graph2Vec, GL2Vec, and LDP do neither utilize the Laplacian matrix nor are capable of considering node attributes which is why those techniques are not modified in this work.

4.5 Clustering Task

After generating whole graph embeddings, we applied the k-means algorithm presented in Section 2.3.1 to the database, initializing the number of clusters to $k = 4$. In the preprocessing step from 4.1.2, we determined the ground-truth data. We used the ground-truth data as a reference for the label assignments of the clustering algorithm for the metrics ARI and AMI. The Davies Bouldin index and the SC conducted the evaluation by the embedding itself. The function that implements the embedding function is illustrated in Algorithm 4.

ALGORITHM 4: EMBEDDING TO CLUSTERING PERFORMANCE (EMBEDDING, GROUND-TRUTH)	
input:	embedding: list of graph embeddings for 14,353 graphs
	ground_truth: ground-truth cluster labels for each product

output: ari, ami, dbp, and silhouette as clustering performance evaluation metrics

```
1  begin
2      y = Kmeans (n_clusters = 4, random_state = 0).fit(embedding)
3      ari = adjusted_rand_score(ground_truth, y.labels_)
4      ami = adjusted_mutual_info_score(ground_truth, y.labels_)
5      dbp = davies_bouldin_score(embedding, y.labels_)
6      silhouette = metrics.silhouette_score(embedding, y.labels_, metric = "euclidean")
7      return ari, ami, dbp, silhouette
```

Algorithm 4. Clustering performance for four different evaluation metrics

5 Evaluation and Discussion

This chapter presents an evaluation of the techniques used and the results of the experiment performed in Chapter 4 which corresponds to the evaluation step of the CRISP-ML diagram in Figure 1. The results are discussed based on the theoretical background of the graph embedding techniques presented in Chapter 2. It starts with a comparison of the techniques using suitable comparison metrics. Subsequently, the performance results for both the graph embedding techniques and the baseline algorithm that performs the clustering task directly on raw data are provided. Finally, the results are discussed and reasons for the performance are stated.

5.1 Comparison of Graph Embedding Techniques

After investigating the notion and the mathematical background of Karate Club's graph embedding techniques, we can derive comparison metrics that affect results of the clustering task. Table 5 illustrates these metrics for all ten graph embedding techniques.

The experiment in Chapter 4 assumes that the default embedding size of the techniques is the most appropriate, as the developers have achieved the best results for these specific values. However, there are techniques such as Graph2Vec and GL2Vec that have a parameter called "dimensions" where the dimensionality of the embeddings is variable and can also be set to 2D or 3D embeddings, but by reducing the size to only a few values, the accuracy of the embeddings suffers, resulting in poor performance.

Another interesting metric is the run time of the experiment for the presented use case in Chapter 3 in seconds. It can be observed that for many techniques the larger the embedding size, the more running time the computer takes to execute the clustering task. The presented run time assumes that the graph creation is already precomputed, and only one graph orientation, either directed or undirected, is considered.

As explained in Chapter 2.1.3, the set of eigenvalues of the Laplacian matrix is referred to as the spectrum of the Laplacian. NetLSD, SF, and IGE exploit the properties of the spectrum of the Laplacian in different ways to obtain whole graph embeddings. The more recent techniques GeoScattering, Feather-Graph, and Wavelet Characteristic use the probability matrix for a random walk to generate graph embeddings. The only technique that uses both methods is IGE, which generates single embedding vectors for each method and then concatenate them.

The last metric in Table 5 reveals which graph features are responsible for the final embedding, which is supposed to represent a whole graph. The graph textualization methods use the WL-kernel and consider both the node label and structural information based on each node's immediate neighbors. While FGSD reconstructs the given graph G based on the pairwise node distances, LDP only considers the node degree information and statistics of the first neighborhood. Most importantly, again, the newer methods GeoScattering, Feather-Graph, and Wavelet Characteristic are able to process any specific vector as feature vector. Therefore, these techniques allow the use of attributed graphs, where node attributes are declared as feature vectors. However,

the aforementioned techniques can also consider other features related to the structure of the graph, such as the logarithmic node degree or the clustering coefficient. The node attributes and the logarithmic node degree can also be concatenated to form a feature vector.

Table 5. Comparison of graph embedding techniques by selected metrics

Technique	**Embedding Size**	**Run Time [s]**	**Spectrum**	**Random Walk**	**Feature Vector**
Graph2Vec	128	180	No	No	Node label and structural information
GL2Vec	128	240	No	No	Node label and structural information
FGSD	200	480	No	No	Pairwise node distances
NetLSD	250	720	Yes	No	Trace of heat or wave kernel
SF	128	600	Yes	No	Spectrum
IGE	652	800	Yes	Yes	Spectrum, probability and commute time for random walk
LDP	160	360	No	No	Node degree information
GeoScattering	N/A	120	No	Yes	Free, logarithmic node degree, clustering coefficient
Feather-Graph	500	90	No	Yes	Free, logarithmic node degree, clustering coefficient
Wavelet Characteristic	1000	900	No	Yes	Free, logarithmic node degree, clustering coefficient

5.2 Results and Discussion

Recall that we are addressing the research question how effective the use of graph embeddings for a clustering task in the manufacturing domain is. Therefore, we must evaluate the accuracy of all ten techniques against a baseline algorithm. In this section, the baseline algorithm is introduced, the results are presented and discussed.

5.2.1 Baseline Algorithm

In the baseline, the clustering task is performed directly with the raw data without the use of graphs. For this purpose, the feature data frame of all 14,353 products was first extracted, manipulated, dimensionally reduced, and finally used as input for the clustering algorithm.

Manipulating the feature data requires two steps: first, replacing all NaN values with zeros, and second, removing the product IDs from the array to prevent misunderstanding of the numerical features. This array of more than 300 features per product was tested as input to the k-means clustering algorithm, which yielded an ARI of about 8.7%. Finally, after applying the widely used tool for dimensionality reduction: Principal Component Analysis (PCA), the algorithm obtained the best results with an ARI of about 10.9% for a number of 100 components.

This work is based on the assumption that the accuracy of the clustering task is significantly higher when using graph embeddings than when using only the raw data.

5.2.2 Results and Discussion

The results are presented for the performance metrics explained in 2.3.2. Each of these metrics captures a different aspect of the task performance. The ARI measures the similarity between ground-truth clustering and clustering of k-means and can be also considered as accuracy. Whereas the AMI is a measure of entropy that describes the uncertainty of a random variable when another variable is known. On the other hand, the SC, which evaluates performance without ground-truth data, calculates whether points are clustered and separated well. Finally, the Davies Bouldin index is defined as the average similarity measure of each cluster with its most similar cluster, taking into account the distances within and between clusters. Since the SC is only defined between -1 and 1, this metric is more accurate than the Davies Bouldin index according to [54].

Table 6 shows the clustering results for all ten graph embedding techniques versus the baseline for undirected and directed graphs considering four different performance metrics. Each cell contains two values, where the result for directed graphs is the value at the top and for undirected graphs the value at the bottom.

Table 6. Clustering results for 10 graph embedding techniques vs. baseline for four different evaluation metrics

Technique	ARI	AMI	Silhouette	Davies Bouldin
Graph2Vec	0.575 0.576	0.468 0.468	0.622 0.668	0.581 0.517
GL2Vec	0.506 -	0.393 -	0.392 -	1.33 -
SF	0.159 0.576	0.200 0.468	0.221 1.00	1.67 7e-6
IGE	0.576 0.501	0.468 0.389	1.00 0.684	1e-11 0.513
NetLSD	0.513	0.394	0.742	0.579

	0.576	0.468	1.00	1e-4
FGSD	0.576	0.468	1.00	0.00
	0.576	0.468	1.00	0.00
LDP	0.576	0.468	1.00	0.00
	0.576	0.468	1.00	0.00
GeoScattering	0.576	0.468	1.00	8e-7
	0.576	0.468	1.00	6e-7
Feather-Graph	0.576	0.468	1.00	2e-5
	-	-	-	-
Wavelet Characteristic	**0.576**	0.468	1.00	6e-5
	0.576	0.468	1.00	6e-5
Baseline (Raw Data)	**0.109**	0.287	0.123	2.45

As Table 6 shows, the performance for the baseline is very poor with lower than 11% for the ARI and approximately 29% for the AMI. The three times better result for the AMI can be argued with the fact that when the ground-truth clustering is known, the uncertainty of a random variable picked from the raw data clustering may be higher than the accuracy for the clustering task represented with the ARI. The graph textualization technique Graph2Vec achieves among others the highest ARI value of around 57% for directed as well as for undirected graphs. This finding can be traced back to the fact that Graph2Vec is also designed for graph clustering and achieved according to [22] a performance of 56% for the malware graph clustering task. However, the performance for unsupervised clustering is rather poor in comparison, which means that the data points of the clusters are rather overlapping and not good separated. There are several possible explanations for such a result.

Since Graph2Vec and GL2Vec initialize graph embeddings randomly and then learn embeddings, the initial graph embedding can limit the values that the algorithm can achieve, leading to fluctuating performance. Then the WL-kernel relabels the nodes with a hash function, where the result may not preserve the structural information about the node very well, which could negatively affect the result for unsupervised performance.

The ARI for SF is about 15% for directed graphs and 57% for undirected graphs. This result helps understanding how important the accurate computation of the Laplacian matrix for the SF algorithm is, because the graph embedding generated by SF consists of the eigenvalues of the normalized Laplacian matrix. The problem here is that for the directed graph model, only the left normalized Laplacian matrix of the outer degree was considered, where a small difference to the actual normalized Laplacian matrix can have significant effects on the graph embedding.

However, the performance for directed graphs of IGE is better than for undirected graphs, although the normalized Laplacian matrix of undirected graphs is replaced by the out-degree left normalized

Laplacian matrix. A possible explanation could be that IGE also uses the random walk matrix, which could be more accurate for directed graphs since there are fewer possible paths within a graph.

Similar to SF, for NetLSD we can observe a rather small difference between the performance of directed and undirected graphs. We should be aware that for NetLSD, the trace of the heat kernel $h_t = tr(H_t) = \sum_j e^{-t\lambda_j}$ is used to calculate the whole graph embeddings. Since the eigenvalues of the normalized Laplacian matrix are used as the exponent of the Euler function, it can be concluded that small differences in the Laplacian matrix can lead to larger differences in the embeddings, which negatively affects the performance.

The last five techniques FGSD, LDP, GeoScattering, Feather-Graph, and Wavelet Characteristic have approximately the same performance for all four metrics. It seems likely that these results arise from the consideration of local node features. FGSD and LDP, for example, do not use the Laplacian matrix or the random walk matrix to generate graph embeddings. The embeddings are calculated based only on the node distances or node degrees of the local neighborhood of a node v_i. Hence, it can be hypothesized that for the given dataset, techniques that focus more on node degree or node distance have higher performance for a clustering task.

This result can be verified by taking a closer look at the newer techniques GeoScattering, Feather-Graph, and Wavelet Characteristic. Since the feature vector for these techniques can be freely chosen, we examined four different feature vectors. Table 7 shows the ARI values for directed graphs for the following feature vectors: 1) node attribute + logarithmic node degree + clustering coefficient 2) node attribute + logarithmic node degree 3) node attribute 4) logarithmic node degree. The performance when considering the node attributes alone or in combination with the logarithmic node degree or the clustering coefficient is lower than when considering the logarithmic node degree itself. It seems likely that the techniques are overwhelmed with node attributes and cannot process the feature values in a meaningful way. Therefore, the use of attributed graphs with the feature value as an attribute is not helpful. The hypothesis stated after discussing the performance of FGSD and LDP can also be applied to these three techniques, since the logarithmic node degree provides information about the local neighborhood of a node v_i.

Table 7. Performance for GeoScattering, Feather-Graph and Wavelet Characteristic using different feature vectors

Technique	**NA-Degree-CC**	**NA-Degree**	**NA**	**Degree**
Geo Scattering	0.576	0.576	0.019	0.576
Feather-Graph	0.322	0.305	0.153	0.576
Wavelet Characteristic	0.003	0.003	0.003	0.576

5.3 Summary

We can summarize the findings of the discussion as following:

- 57% ARI performance and 100% for the SC for both directed and undirected graphs are achieved only with FGSD, LDP, GeoScattering, Feather-Graph, and Wavelet Characteristic.
- Major influence on the performance has the consideration of the local neighborhood by node distance or node degree.
- Although 57% is not an excellent score for the application, it clearly outperforms the baseline that only achieves around 11% ARI.
- Considering feature values as node attributes in GeoScattering, Feather-Graph, and Wavelet Characteristic is contra-productive and leads to poor performance because the algorithm is overloaded.
- Karate Club's graph embeddings are customized for specific datasets and need to be modified for competitive results in the manufacturing domain.
- No direct conclusion whether a model with directed graphs or a model with undirected graphs is better, since it depends on the embedding technique, in some cases better in others worse.
- Only numerical data has been processed, progress must be made on strings as usual in KG.
- It was assumed that the clustering task is easier than the classification task since the dataset is imbalanced. However, most of Karate Club's whole graph embeddings are tested only for classification, so the next step is classification.

6 Conclusion and Outlook

Finally, this chapter summarizes and discusses the key takeaways of this work. Furthermore, an outlook is given on the need for additional research on using graph embeddings for downstream ML tasks in the manufacturing domain.

6.1 Conclusion

The goal of this work was to understand how effective the use of graph embeddings is for a clustering task in the manufacturing domain. Based on the literature review and data exploration of the use case, the development and training of the ten graph embedding techniques followed. The literature review revealed that the graph embedding techniques were designed and tested by Karate Club specifically for benchmark datasets, which include social media platforms and chemical or biochemical structures. The techniques assume that the graphs are undirected and unattributed, and most of the techniques are tested for a graph classification task. In this work, instead of graph classification, we perform graph clustering to test how much information the graph embeddings contain and whether they can be used to identify different product variants. This would be a "preprocessing" step in a classification pipeline. The fact that the measured production data are highly unbalanced and that a significant number of NaN values are included in the data are further reasons for performing clustering. Since we know the entities of the manufacturing system, namely line, station, and feature, we can conclude that a directed graph would better reflect the relationship between these entities than an undirected graph. Therefore, the graph model was developed for both directed and undirected graphs.

Since the task was to test ten graph embedding techniques, we reduced the complexity of the problem by creating a smaller dataset using data from only four different product variants. The data was preprocessed to prepare it for graph creation. Depending on the specific graph embedding technique, different steps were performed prior to graph creation. The graph embedding techniques originally proposed by Karate Club were either modified to be able to create directed graphs or to take into account node attributes. Subsequently, graph embeddings for each of these ten techniques were created for both undirected and directed graphs. After that, the k-means clustering algorithm, initialized with a number of $k = 4$ clusters, is used to assign the generated graph embeddings to clusters. To evaluate the performance of the clustering, the ground-truth data is used as a reference.

Finally, the performance of the ten techniques was evaluated for undirected and directed graphs using a baseline algorithm that directly clusters the raw data. Previously, the ten graph embedding techniques were compared using selected metrics to better explain the results later. Four different metrics that capture different similarity information of the products were used for the evaluation. The results show that an ARI of 57% can be achieved with multiple graph embedding techniques for both directed and undirected graphs. Although 57% is not an excellent score for clustering, the created graph embedding techniques outperform the baseline, which only achieves around 11%

ARI. It was observed that techniques that consider information about the local neighborhood of a node, such as node degree or node distance, achieved significantly higher performance scores than others. Considering node attributes resulted in poor performance due to algorithm overload. This work presented a first approach to extract meaningful features from KG for a clustering task. The problem is that Karate Club's graph embedding techniques are customized to specific datasets and need to be redesigned to produce more competitive results for the use in the manufacturing domain. In conclusion, the current version of whole graph embedding techniques developed by Karate Club can be used for a clustering task on the Bosch production line dataset. However, the results are not very promising for use in production, which is why we need to conduct further research, as explained in the Outlook.

6.2 Outlook

First, the only graph embedding technique tested for a graph clustering task is Graph2Vec. All ten techniques were successfully tested for a graph classification task. For this reason, further research dealing with Karate Club's graph embeddings should perform graph classification instead of graph clustering.

The whole graph embedding techniques of Karate Club are designed for processing ordinary graphs instead of KG. This is the reason why only numerical data can be processed by the given techniques, but KG usually consist of both textual and numerical information, and in this work the textual information has not been considered. Therefore, further research on special KG embeddings, such as those developed by [55] or [56], should be conducted to process all the information obtained from the manufacturing system.

Ideally, one of the newer and stronger performing algorithms such as GeoScattering, Feather-Graph, or Wavelet Characteristic should be combined with a KG embedding, where the design of this new algorithm should be suited to the given dataset. When the performance of this algorithm is very competitive, the model can be deployed to production as shown in the CRISP-ML diagram in Figure 1. After deployment, it should be assured that the performance of the system is monitored regularly, and the production line is also maintained.

Bibliography

[1] Google Cloud (2021): Artificial Intelligence acceleration among manufacturers, URL: https://services.google.com/fh/files/blogs/google_cloud_manufacturing_report_2021.pdf [Accessed: 08/04/2022].

[2] Schume, Philipp (2020): Improve product quality and yield with intelligent, secure, and adaptable manufacturing operations - Industry 4.0 brings opportunities to infuse AI into manufacturing, *IBM Business Operations Blog*, URL: https://www.ibm.com/blogs/internet-of-things/iot-manufacturing-ready/ [Accessed: 08/04/2022].

[3] Needham, Mark; Hodler, Amy E. (2019): Graph Algorithms: Practical Examples in Apache Spark and Neo4j, O'Reilly Media, Sebastopol, Beijing, Boston, Farnham, Tokyo, 2019.

[4] Cassoli, Beatriz B.; Jourdan, Nicolas; Nguyen, P. H. et al. (2021): Frameworks for data-driven quality management in cyber-physical systems for manufacturing: A systematic review, In: 15th CIRP Conference on Intelligent Computation in Manufacturing Engineering - CIRP ICME '21 Virtual Conference.

[5] Cassoli, Beatriz B.; Jourdan, Nicolas; Metternich, Joachim (2022): Knowledge Graphs For Data And Knowledge Management In Cyber-Physical Production Systems, In: 3rd Conference on Production Systems and Logistics.

[6] Studer, Stefan; Bui, Thanh B.; Drescher, Christian et al. (2021): Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology, In: Machine Learning and Knowledge Extraction, vol. 3, no. 2, pp. 392–413.

[7] Diestel, Reinhard (2017): Graph Theory, Berlin (Springer), 2017.

[8] Bondy, J. A.; Murty, U. S. R. (1976): Graph theory with applications, New York (North-Holland), 1976.

[9] Husen, William: Random Walks on Graphs, In: Linear Algebra with Applications. Ohio State University.

[10] Spielman, Daniel A. (2018): Random Walks on Graphs, In: Spectral Graph Theory. Yale University.

[11] Oktatas, Tony (2011): Digraphs. Eötvös Lorand University.

[12] George, Alan; Gilbert, John R.; Liu, Joseph W.H. (1993): Graph Theory and Sparse Matrix Computation, New York (Springer), 1993.

[13] Beineke, Lowell W.; Wilson, Robin J.; Cameron, Peter J. (2004) Topics in algebraic graph theory, Cambridge (Cambridge Univ. Press), 2004.

[14] Sparse Matrices (scipy.sparse), URL: https://docs.scipy.org/doc/scipy/reference/sparse.html [Accessed: 08/04/2022].

[16] Edding, Matt (2019): Sparse Matrices, *Python and Data Science Blog*, URL: https://matteding.github.io/2019/04/25/sparse-matrices/ [Accessed: 08/04/2022].

[17] Chung, Fan R. K. (1997): Spectral graph theory, In: CBMS Conference on Recent Advances in Spectral Graph Theory, Providence (American Mathematical Society), 1997.

[18] Mondaini, Rubem P. (2021): Trends in Biomathematics: Chaos and Control in Epidemics, Ecosystems, and Cells, Cham (Springer), 2021.

[19] NVIDIA Data Science Glossary (2022): NetworkX, URL: https://www.nvidia.com/en-us/glossary/data-science/networkx/ [Accessed: 07/22/2022].

[20] Hagberg, Aric A.; Schult, Daniel A.; Swart, Pieter J. (2008): Exploring Network Structure, Dynamics, and Function using NetworkX, In: Proceedings of the 7th Python in Science Conference, p.11-15, Pasadena, CA, 2008.

[21] Rozemberczki, Benedek; Kiss, Oliver; Sarkar, Rik (2020): Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs, In: Proceedings of the 29th ACM International Conference on Information and Knowledge Management, 2020.

[22] Narayanan, A.; Chandramohan, M.; Venkatesan, R. et al. (2017): graph2vec: Learning Distributed Representations of Graphs, In: arXiv: 1707.05005, 2017.

[23] Tsitsulin, A.; Mottin, D.; Karras, P. et al. (2018): NetLSD: Hearing the Shape of a Graph, In: arXiv: 1805.10712, 2018.

[24] Wang, L.; Huang, C.; Ma, W. et al. (2021): Graph Embedding via Diffusion-Wavelets-Based Node Feature Distribution Characterization, In: arXiv: 2109.07016, 2021.

[25] Gedeon, T.; Wong, K. W.; Lee, M. (2019): Neural Information Processing, Cham (Springer), 2019.

[26] Shervashidze, N.; Schweitzer, P.; van Leeuwen, E. J. et al. (2011): Weisfeiler-Lehman Graph Kernels, In: Journal of Machine Learning Research, vol. 12, no. 77, pp. 2539–2561, 2011.

[27] Le, Quoc V.; Mikolov, Tomas (2014): Distributed Representations of Sentences and Documents, In: International Conference on Machine Learning, pp. 1188–1196, 2014.

[28] Mikolov, T.; Sutskever, I.; Chen, K. et al. (2013): Distributed Representations of Words and Phrases and their Compositionality, In: arXiv: 1310.4546, 2013.

[29] Verma, S.; Zhang, Z.-L. (2017): Hunt For The Unique, Stable, Sparse And Fast Feature Learning On Graphs, In: Advances in Neural Information Processing Systems, vol. 30, 2017.

[30] Lara, N. d.; Pineau, E. (2018): A Simple Baseline Algorithm for Graph Classification, In: arXiv: 1810.09155, 2018.

[31] Shuman, D. I.; Ricaud, B.; Vandergheynst, P. (2013): Vertex-Frequency Analysis on Graphs, In: arXiv: 1307.5708, 2013.

[32] Zhang, Y.; Xiong, Y.; Kong, X et al. (2021): IGE+: A Framework for Learning Node Embeddings in Interaction Graphs, In: IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 3, pp. 1032-1044, 2021.

[33] Cai, C.; Wang, Y. (2018): A simple yet effective baseline for non-attributed graph classification, In: arXiv: 1811.03508, 2018.

[34] MATLAB & Simulink - MathWorks Deutschland (2020): What is a Wavelet?, URL: https://de.mathworks.com/help/wavelet/gs/what-is-a-wavelet.html [Accessed: 07/15/2022].

[35] MATLAB & Simulink - MathWorks Deutschland (2020): Choose a Wavelet, URL: https://de.mathworks.com/help/wavelet/gs/choose-a-wavelet.html, [Accessed: 07/15/2022].

[36] Gao, F.; Wolf, G.; Hirn, M. (2018): Geometric Scattering for Graph Data Analysis, In: Proceedings of the 36th International Conference on Machine Learning, pp. 2122–2131, 2019.

[37] Bruna, J.; Mallat, S. (2013): Invariant scattering convolution networks, In: IEEE transactions on pattern analysis and machine intelligence, vol. 35, no. 8, pp. 1872–1886, 2013.

[38] Rozemberczki, B.; Sarkar, R. (2020): Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models, Conference on Information and Knowledge Management 2020, 2020.

[39] Rozemberczki, B.; Allen, C.; Sarkar, R. (2021): Multi-scale Attributed Node Embedding, In: Journal of Complex Networks, vol. 9, no. 2, 2021.

[40] Donnat, C.; Zitnik, M.; Hallac, D. et al. (2017): Learning Structural Node Embeddings via Diffusion Wavelets, In: arXiv: 1710.10321, 2017.

[41] Pedregosa, F.; Varoquaux, G.; Gramfort, A. et al. (2011): Scikit-learn: Machine Learning in Python, In: Journal of Machine Learning Research, 2011.

[42] Kersting, Kristian (2019): Support Vector Machines and Unsupervised Learning, In: Machine Learning Applications. Technical University of Darmstadt.

[43] Rand, W. M. (1971): Objective Criteria for the Evaluation of Clustering Methods, In: Journal of the American Statistical Association, vol. 66, no. 336, p. 846, 1971.

[44] Gates, A. J.; Ahn, Y.-Y. (2017): The Impact of Random Models on Clustering Similarity, In: Journal of Machine Learning Research 18 (87), 2017.

[45] Gray, R. M. (2011): Entropy and Information Theory, Boston (Springer), MA, 2011.

[46] Cover, T. M.; Thomas, J. A. (2005): Elements of Information Theory, Wiley, 2005.

[47] Danyluk, A. (2009): Proceedings of the 26th Annual International Conference on Machine Learning, ACM, New York, NY, 2009.

[48] Bosch Global (2022): Company Overview, URL: https://www.bosch.com/company/ [Accessed: 5/31/2022].

[49] Kaggle (2016): Bosch Production Line Performance, URL: https://www.kaggle.com/competitions/bosch-production-line-performance/ [Accessed: 5/31/2022].

[50] David, F. N.; Tukey, J. W. (1977): Exploratory Data Analysis, Biometrics, vol. 33, no. 4, p. 768, 1977.

[51] Pradhan, Debasish (2022): Bosch production line performance, *Medium*, URL: https://medium.com/@pradhandebasish2046/bosch-production-line-performance-166aa65629c1 [Accessed: 08/04/2022].

[52] Mangal, A.; Kumar, N. (2016): Using Big Data to Enhance the Bosch Production Line Performance: A Kaggle Challenge, In: arXiv: 1701.00705, 2016.

[53] Broeren, P. M. T. (2006): Automatic Autocorrelation and Spectral Analysis, London (Springer), 2006.

[54] Petrovi, S.: A Comparison Between the Silhouette Index and the Davies-Bouldin Index in Labelling IDS Cluster, In: Proceedings of the 11th Nordic workshop of secure IT systems, pp. 53-64.

[55] Dai, Y.; Wang, S.; Xiong, N. N. et al. (2020): A Survey on Knowledge Graph Embedding: Approaches, Applications and Benchmarks, Electronics, vol. 9, no. 5, p. 750, 2020.

[56] Wang, Q.; Mao, Z.; Wang, B. et al. (2017): Knowledge Graph Embedding: A Survey of Approaches and Applications, In: IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 12, pp. 2724–2743, 2017.

[57] Hazewinkel, M. (2002): Encyclopaedia of mathematics, Berlin (Springer), 2002.

[58] Sebbar, A.; Sebbar, A. (2012): Equivariant functions and integrals of elliptic functions, Geometriae Dedicata, vol. 160, no. 1, pp. 373–414, 2012.