Master Thesis

Gaining Customer Insights using Machine Learning on Graphs

University of Basel

Author:
Michael von Siebenthal

Supervisor:
Prof. Dr. Dietmar Maringer

July 2, 2021

Abstract

The abstract comes here

Declaration

"I hereby declare - that I have written this master thesis without any help from others and without the use of documents and aids other than those stated in the references, - that I have mentioned all the sources used and that I have cited them correctly according to the established academic citation rules, - that the topic or part of it are not already the object of any work or examination of another course unless explicitly stated,- that I am aware of the consequences of plagiarism at the Business and Economics Faculty of University of Basel."

Michael von Siebenthal, Martikel-Nr.: 2015-256-837, Date: July 2, 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The aim of this thesis is to explore the relatively new field of machine learning on graphs/networks. In particular, this thesis will focus on graph machine learning applications for the purpose of gaining customer insights. Graph machine learning is a current frontier in machine learning and has vast applications in many areas as recently shown by the success of AlphaFold (Senior et al. 2020). AlphaFold made a breakthrough for predicting protein structures where they made use of the observation that a folded protein can be considered as a spatial graph (AlphaFold 2020). In addition, there is a vast range of applications for graphs in the fields of natural science, social science and many more as shown by the excellent overview given by Zhou et al. (2020). In principle, graphs are useful whenever one wants to consider the interactions between observations. Graph machine learning has therefore become a promising field as it allows for the consideration of standard observational data as well as any prevalent interactions or relationships between the observations. The field of graph machine learning focuses on the developing methods which can exploit this richer information for various tasks such as (customer-) classification. To provide a better understanding on how graphs in general as well as graph machine learning are relevant to business & economics related fields such as gaining customer insights, some motivating examples are provided in the following section.

1.1 Relevance to Economics

From a business & economics perspective, graphs are particularly interesting if one wants to for instance model the interactions between institutions. An example for this is a study published by Schweitzer et al. (2009) which created a graph showing the interdependencies or interactions of international banks as a network shown in figure 1.1. This is a useful representation of such interdependencies and provides important information for making the banking system more robust and resilient.
Another interesting application of graphs for business & economics are social inter-

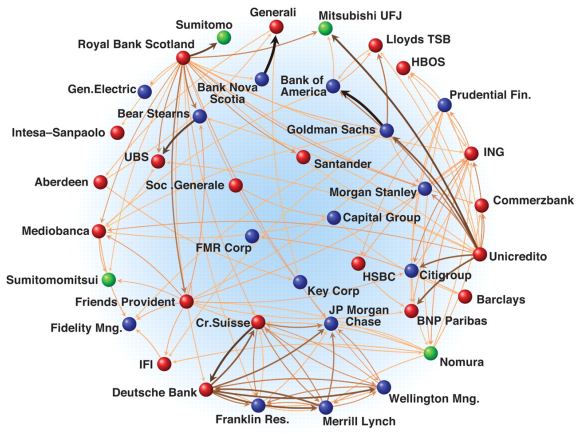

Figure 1.1: Bank Network
(Schweitzer et al. 2009, p. 424)

actions. While there are many different interesting applications where social interactions are of interest, the focus for this thesis is placed on gaining customer insights for the purpose of improving products & services as well as marketing. Indeed, this is one of the main areas where social networks such as Facebook or search providers like Google make their revenue by providing consumer insights or selling targeted advertising (Facebook 2021, Alphabet 2021). Both Facebook and Google have the advantage, that their businesses naturally capture relational or more generally network data which can be represented as graphs. Most researchers or companies however do not have access to such data. Companies for instance may have access to large amounts of customer data, however they typically would not have access to relational information (e.g. which client is connected with which other clients). The same is true for researchers, where social scientists often collect data via anonymous surveys which make the collection of network data basically impossible. In most cases this means, that companies or researchers can only generate- or have access to data which does not contain relational information such as social interactions or connections. It is important to highlight, that there is a lot of social network data available online. This network data however typically only contains the network structure/connections. The feature data such as demographic data or topic specific variables of interest of the people present in the network is however typically not provided. Without any feature data, graphs provide rather limited information and in many cases are not useful for tasks such as providing customer insights. This is an issue in terms of data access and is a frustrating reality which also affects this thesis. It however also motivated the research question which will be presented in the following section.

1.2 Research Topic

The difficult access to graph data which included feature information, motivated the search for alternatives. A review of the literature revealed, that a form of synthetic graph generation could provide a solution to the data access problem. Classic graph generation procedures include the famous Erdös-Rényi graphs (Erdös & Rényi 1959), the small-world model by Watts & Strogatz (1998), the well-known model by Barabási & Albert (1999) and more recently Kronecker Graphs by Leskovec et al. (2010). These models are all very instructive regarding the graph generation process and for understanding graph properties. These networks however all have the shortcoming, that they do not allow for assigning feature data to the network. It became clear, that one would have to find or develop a method which makes use of feature data for the graph generation process. Fortunately, such a method was developed by Kim & Leskovec (2012) which introduced Multiplicative Attribute Graphs (MAG). This model uses feature data which is referred to as attribute data by the authors to generate graphs and appears to yield the desired result of graph with feature data. This model and its specifications will be introduced in detail in the theory section.

More recently, researchers have focused their attention to generative graph models. These models create graphs with features using real graphs as a training input. Examples for such models are Graph Recurrent Neural Networks (You et al. 2018) and deep generative graph models (Li et al. 2018). These are very fascinating models which can be used to recreated or scale graph data. For the purpose of this thesis, these methods were not purposeful, as it requires existing graph data to be present. For this thesis this unfortunately not the case. Nevertheless, this is an interesting current topic for graph generation.

The access problem to graph data therefore might be resolved using the MAG method. This method is interesting as most researchers or companies have access to large amounts of feature data such as customer databases or feature data which can be created using well-known methods like survey. It could therefore be of interest to generate a semi-synthetic graph with the MAG method using more available non-graph data. Semi-synthetic refers to the fact, that the synthetic graph is generated using real collected feature data. Therefore it is referred to as semi-synthetic and differs to the fully synthetic methods previously outlined. The hope is, that relational information generated in the graph would provide useful additional information for machine learning task such as gaining customer insight. The aim of the thesis is therefore to investigate to what extent semi-synthetic graphs are useful for graph based machine learning techniques. Of course, semi-synthetic graphs cannot replace

real graphs and graph machine learning on semi-synthetic graphs is unlikely to perform as well as it would on real graphs. It is however possible, that graph machine learning on semi-synthetic graphs is competitive, if not superior to machine learning methods which are not graph based and can only consider feature data. In order to assess the viability of this approach, the results using graph machine learning will be compared to standard machine learning approaches. To provide a better overview, a high level overview of machine learning is provided in the following section. To close this section, the research question is presented formally as follows:

Research Question:

To what extent are semi-synthetic graphs based on real feature data useful for machine learning?

1.3 Overview Machine Learning

This section will provide a high-level overview of machine learning and will specify the type of machine learning task used in this thesis. To start, it is important to correctly categorize machine learning. There are many related big topics such as data science, big data or artificial intelligence and it is often not clear what exactly is meant. A good graphical overview was generated by the Frauenhofer Institut and is shown in figure 1.2.

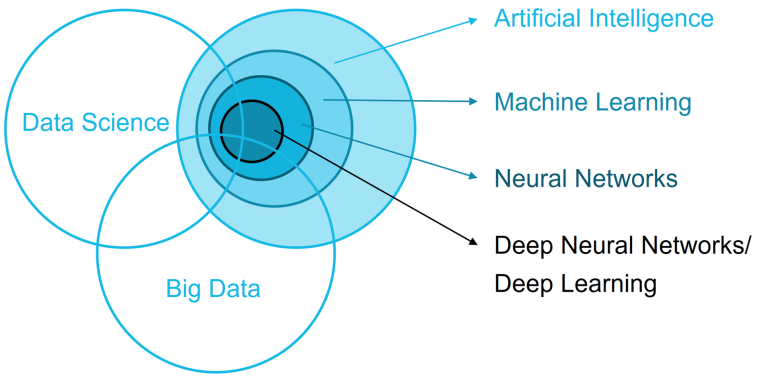

Figure 1.2: Overview Machine Learning
Frauenhofer Institut (2021)

Figure 1.2 shows well, how these different terms are related with each other. Machine learning in particular, is mostly ascribed to the domain of artificial intelligence, it however also shares its domain with data science and big data. It is thus at the intersection of these three main domains. Machine learning techniques such as neural networks and deep neural networks are specific methods within machine

learning and are often referred to separately due to their popularity. In this thesis, differentiating between machine learning and neural networks is not necessary as all considered machine learning methods will be used for the same task. Machine learning can be applied for various tasks and is again best presented visually in figure 1.3.

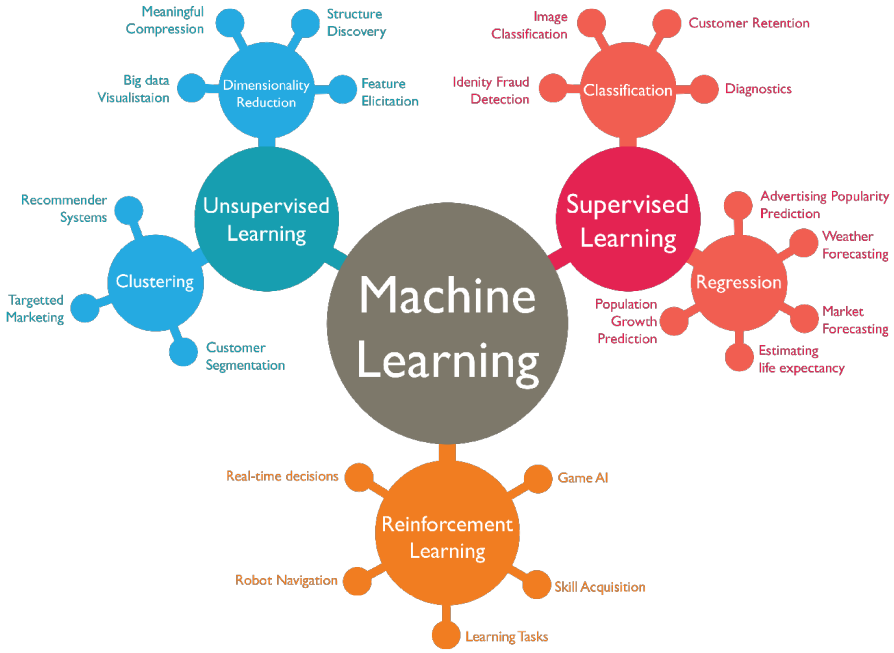

Figure 1.3: Overview Machine Learning Tasks
Artisan's Asylum (2020)

It is shown in figure 1.3 that the main tasks for machine learning involve classification, regression, reinforcement learning, clustering and dimensionality reduction. This thesis will focus on classification tasks. This task was chosen given the available data for this thesis and because it allows for a nice comparison of different methods. Well known standard machine learning methods for classification tasks include logistic regression (Cramer 2002), naive bayes (Zhang 2004), support vector machines (Platt et al. 1999), random forest classifiers (Breiman 2001), AdaBoost (Freund & Schapire 1997) and artificial neural networks (McCulloch & Pitts 1943). This is an incomplete list of popular machine learning methods that can be used for classification tasks. Classifications tasks can be applied for various settings such as predicting whether a customer is satisfied or whether to grant a mortgage to a client among many others. The aforementioned machine learning methods have in common, that they all only consider feature data from a customer database or from a survey.

Graph machine learning methods are different in that they consider both feature data as well as the connections present in a graph. If one wants to categorize

graph machine learning within the framework shown in figure 1.2, it is probably best categorized as a special form of a neural network. It is however not necessarily a deep neural network, as network depth does not necessarily improve the model and can be even contra-productive. Within graph machine learning, there are two main approaches. The first approach focuses on learning vector representations of graphs which are subsequently used for downstream machine learning tasks using the standard machine learning methods previously presented. This approach includes methods such as DeepWalk (Perozzi et al. 2014) or Node2Vec (Grover & Leskovec 2016) among others. The second approach involves the application of graph neural networks of which their exist many different approaches. These approaches include methods such as graph convolutional networks (Kipf & Welling 2016) or GraphSage (Hamilton et al. 2017) and many more.

The required theoretical background for understanding graph machine learning, graph generation and graphs in general is provided in the following chapter.

Chapter 2

Theory

This chapter will cover the necessary theoretical background and consists of the following main parts:

1. Graph Theory
2. Machine Learning on Graphs
3. Graph Generation

2.1 Graph Theory

This section provides a brief introduction to graph theory, with a focus on the relevant aspects for this master thesis. The theory presented is primarily taken from the book "Networks: An Introduction" by Mark Newman (2010).

Graph theory is an old field of mathematics and can be traced back to Leonhard Euler and the famous "Königsberg Bridge Problem" (Euler 1736). The study of graphs has had a recent revival thanks to its useful applications in areas such as the Google algorithm PageRank Page et al. (1999) and machine learning. Graphs are special data structures as shown in Figure 2.1. The terms graph and network are often used interchangeably and have identical meaning for the purpose of this master thesis. Typically, the term graph is used more commonly when referring to mathematical analysis of graphs and the term network is more commonly used for data science purposes.

The graph shown in Figure 2.1 corresponds to an undirected graph in which the connections between the vertexes are mutual. In a directed graph for instance, vertex A could be connected to vertex B, however vertex B need not be connected to vertex A. For the purpose of this thesis, only undirected graphs are considered.

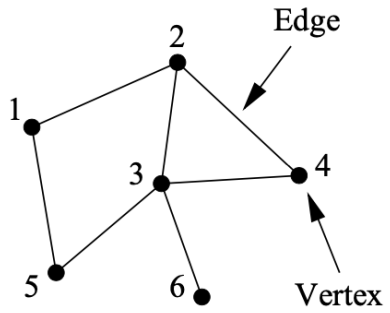

Figure 2.1: Example of a Graph
(Newman 2010, p. 111)

Vertexes are often referred to as nodes and the terms will be used interchangeably. Edges refer to the connections between the vertices. Edges are often also referred to as links and the terms will be used interchangeably as well. Graphs may have additional elements such as multi-edges or self-edges. Self-edges refer to nodes which have a looped link to itself. This can be considered as a feedback loop of a node on to itself. Lastly, multi-edges refer to direct node connections with multiple paths.

In terms of mathematical notation, graphs are typically defined as follows:

$$G(V, E) \tag{2.1}$$

G refers to the graph as an output. V refers to the set of vertices present in the graph and E refers to edges present between the vertices.

Adjacency Matrix

The adjacency matrix A is defined as a $n \times n$ matrix, where n refers to the number of vertices present in the graph. Each vertex is therefore recorded by a column and a row in the adjacency matrix. The elements in the adjacency matrix are further typically defined as follows:

$$A_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ and } j \text{ are connected by an edge} \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

For illustration, the adjacency matrix of the graph shown in Figure 2.1 is shown as follows:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

As one can see, if vertex i and j are connected, this is recorded with 1 and 0 otherwise. Note, that all the elements on the $diag(A)$ are equal to 0. This is because there are no self-edges present in figure 2.1. Nodes with self-loops would have a 1 recorded on the corresponding diagonal element of the adjacency matrix. As this is an undirected network, the adjacency matrix is symmetric. There are many additional aspects one could mention with regard to the adjacency matrix, they are however not relevant for this thesis. For additional information regarding the adjacency matrix, the book "Networks: An Introduction" by Mark Newman (2010) is highly recommended.

Degree Measures

An important measure for graphs are the degrees denoted by k of the vertices. Degrees refer to the number of edges connected to a vertex. The degrees of vertex denoted by i can be formulated as (Newman 2010, p. 133):

$$k_i = \sum_{j=1}^{n} A_{ij} \tag{2.3}$$

For an undirected graph, edges have two ends. This is due to the fact that vertices connected by an edge are mutually connected. In terms of the sum of the degrees of all vertices, we can therefore write for a graph with m edges (Newman 2010, p. 133):

$$2m = \sum_{i=1}^{n} k_i \tag{2.4}$$

The sum of all degrees is therefore just the number of edges m multiplied by 2. In terms of statistical measures, the mean degree c of a vertex is defined as follows (Newman 2010, p. 134):

$$c = \frac{1}{n} \sum_{i=1}^{n} k_i = \frac{2m}{n} \tag{2.5}$$

In order to define the connectance or density of a graph, we must first observe, that

the maximum number of edges is given by (Newman 2010, p. 134):

$$\binom{n}{2} = \frac{1}{2}n(n-1) \tag{2.6}$$

The density ρ can therefore be written as (Newman 2010, p. 134):

$$\rho = \frac{m}{\binom{n}{2}} = \frac{2m}{n(n-1)} = \frac{c}{n-1} \tag{2.7}$$

Note, that the density ρ lies strictly between $0 \leqslant \rho \leqslant 1$. In addition, for sufficiently large graphs, one can approximate $\rho = \frac{c}{n}$.

Eigenvector Centrality

The degrees of a vertex shown in the previous section already correspond to the simplest form of centrality measures. The issue with this measure however is, that the every neighbor of vertex i are valued the same. This is a problem, as not all neighbors are of equal importance due to:

1. Number of neighbors
2. Importance of neighbor
3. both

There are many different alternative centrality measures which can consider the factors listed above such as eigenvector centrality, Katz centrality or PageRank (Katz 1953, Page et al. 1999, Landau 1895, Newman 2010). As we are only dealing with simple undirected graphs, eigenvector centrality will suffice, where the other mentioned methods are adaptations to the eigenvector centrality.

Eigenvector centrality gives all vertices a score proportional to the sum of the scores of its neighbors. This is a procedure in which typically the initial centrality x_i of vertex i is guessed to be $1 \; \forall \; i$. This can be used to calculate the centralities of the neighbors of i which is denoted as x'_i. We can thus write (Newman 2010, p. 169):

$$x'_i = \sum_j A_{ij} x_j \tag{2.8}$$

In matrix form:

$$x' = Ax \tag{2.9}$$

This process is repeated t times to provide better estimates (Newman 2010, p. 170):

$$x(t) = A^t x(0) \tag{2.10}$$

Where $x(0)$ is a linear combination of (Newman 2010, p. 170):

$$x(0) = \sum_i c_i v_i \tag{2.11}$$

v_i corresponds to the eigenvectors of the adjacency matrix A and c_i corresponds to an appropriately chosen constant. Therefore we can write (Newman 2010, p. 170):

$$x(t) = A^t \sum_i c_i v_i = \sum_i c_i k_i^t v_i = k_i^t \sum_i c_i \left[\frac{k_i}{k_1}\right]^t v_i \tag{2.12}$$

In the above equation, k_i correspond to the eigenvalues of A, the adjacency matrix. k_1 corresponds to the largest eigenvalue of A. As $\frac{k_i}{k_1} < 1, \ \forall \ i \neq 1$, the term is decaying as $t \to \infty$. The centralities x can therefore be written as fulfilling following condition (Newman 2010, p. 170):

$$Ax = k_1 x \tag{2.13}$$

Lastly, the eigenvector centrality is defined as (Newman 2010, p. 170):

$$x_i = k_1^{-1} \sum_j A_{ij} x_j \tag{2.14}$$

Closeness Centrality

Closeness centrality, C_i, is defined as the average distance from a vertex to the other vertices. This centrality measure is defined as follows (Newman 2010, p. 182):

$$C_i = \frac{1}{l_i} = \frac{n}{\sum_j d_{ij}} \tag{2.15}$$

In this measure, central vertices exhibit high closeness centrality and are therefore closer connected to other vertices compared to vertices with low closeness centrality. l_i refers to the average of the geodesic distances d_{ij} of vertex i.

Betweenness Centrality

This centrality measures to which extent a vertex lies on paths between other vertices. For instance, a bottle neck vertex would exhibit a large betweenness centrality as many, if not all nodes must pass through it. More formally, betweenness centrality, x_i, is defined as (Newman 2010, p. 187):

$$x_i = \sum_{st} \frac{\eta_{st}^i}{g_{st}} \tag{2.16}$$

In the above equation, η_{st}^i refers to the number of geodesic paths from s to t which pass through vertex i. Further, g_{st} is defined as the number of geodesic paths between vertex s and t.

In order to allow for better comparison of betweenness centrality, it is often standardized by the number of connected vertex pairs s and t denoted as η^2. The betweenness centrality can therefore be expressed as (Newman 2010, p.190):

$$x_i = \frac{1}{\eta^2} \sum_{st} \frac{\eta_{st}^i}{g_{st}} \tag{2.17}$$

With this measure, the betweenness centrality is within the range $0 \leqslant 1$

2.2 Machine Learning on Graphs

Graph structures are special in that the data points in a graph have connections with each other. A practical example for this are social networks. In a social network, the profiles of "Peter" and "Paul" might be connected because "Peter" and "Paul" are friends. In addition, "Paul" and "Peter" can only ever reach each-other if they are directly or perhaps indirectly connected. This aspect is unique to graph or network data and provides both additional information as well as additional complexity to graph data. This property does not allow for comparing nodes in a graph in terms of euclidean distances as only connected nodes can reach each-other. For the purpose of this thesis, machine learning on graphs will be categorized into two categories:

1. Graph representation learning with subsequent downstream machine learning
2. Graph neural networks

Graph representation learning refers to models in which node embeddings of a graph are learned. Specifically, this approach yields vector representations of the nodes in a graph for which distance measures such as Euclidean distances can be measured. These node vector representations can then be used as feature data for standard machine learning methods. Graph neural networks can perform both graph representation learning as well as machine learning tasks within the same model/step. This combination is what makes graph neural networks stand out compared to other graph representation learning methods.

In the following subsections, the theory for graph representation learning models as well as graph neural networks are introduced.

2.2.1 Graph Representation Learning

The aim of graph representation learning is to generate node embeddings in the form of a d-dimensional vector representation. The resulting vector embeddings can then be used for "standard" machine learning applications. A graphical representation of this task is shown in figure 2.2.

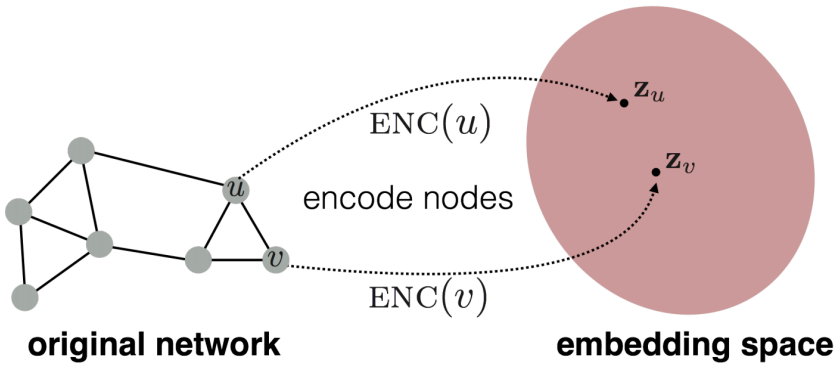

Figure 2.2: Network Embedding
Leskovec (2021)

To generate node embeddings of a graph, one has to define an encoder which transforms nodes in a graph into their embedding space as shown in figure 2.2. The nodes must be embedded in such a manner that similar nodes in the graph are also embedded closely in the embedding space. A common measure for similarity is to find vector embeddings z of nodes u and v such that (Leskovec 2021):

$$z_u^T z_v \approx similarity(u, v) \tag{2.18}$$

The dot product of the two node embedding vectors should thus approximately equal the similarity of the corresponding nodes in the graph. There are different approaches for defining node similarity. Graph factorization was introduced as an early solution (Ahmed et al. 2013). More recent and more successful approaches include methods which make use of random walks. In the context of random walks, similarity is defined as (Leskovec 2021):

$$z_u^T z_v \approx \text{Probability that node } u \text{ and } v \text{ co-occur on a random walk over the graph} \tag{2.19}$$

The methods DeepWalk (Perozzi et al. 2014) and its generalization Node2Vec (Grover & Leskovec 2016) successfully apply the similarity measure shown in equation 2.19.

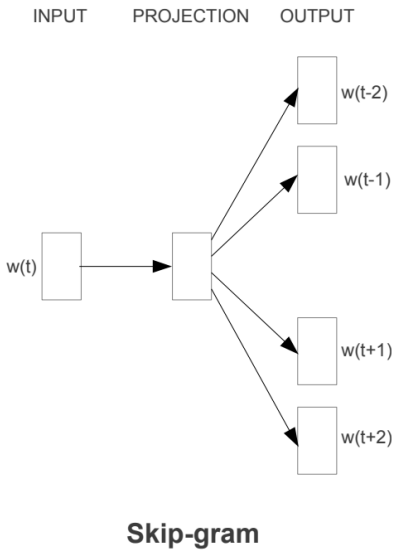

Figure 2.3: Skip-Gram Architecture
(Mikolov et al. 2013a, p. 5)

Another noteworthy method called LINE (Tang et al. 2015) also makes use of random walk co-occurrences as its similarity measure. In order to remain focused, only DeepWalk and Node2Vec are considered for this thesis. These two models are well suited for the given task and are among the most popular graph representation learning methods.

DeepWalk and Node2Vec make use of methods which have its origin in natural language processing (NLP). Specifically they makes use of the Skip-Gram model introduced by Mikolov et al. (2013a,b). The Skip-Gram model is a core component of DeepWalk and Node2Vec, which is why it is explained in detail before proceeding to DeepWalk and Node2Vec.

In NLP words are one-hot encoded as inputs for the Skip-Gram model which learns vector representations of the input words. The aim of the Skip-Gram model is then to predict the context of the input word by predicting its neighboring words in a sentence. A basic overview of the Skip-Gram model is provided in figure 2.3.
The basic layout shown in figure 2.3 depicts the high level procedure of the Skip-Gram model. To make this model more specific, the input corresponds to the one-hot encoded vector at row t of the input matrix W with dimensions $T \times T$, where every row corresponds to a one-hot encoded word. $W(t)$ is linearly passed to the projection, which involves calculating the dot product of $W(t)$ with the weight matrix Φ which has dimensions $T \times D$. D refers to the number of dimensions

which are to be included in the projection vector h and is a hyper parameter. The projection vector h is then linearly passed again with another weight matrix Ψ which has dimensions $D \times T$. This creates the output vector u which is then used to predict the correct context word c from the vocabulary of C number of context words. To do so, the training target is set to maximize the average log probability of the correct context word for every input word w_t. More formally (Mikolov et al. 2013b, p. 2):

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leqslant j\leqslant c, j\neq 0}\log p(w_{t+j} \mid w_t) \tag{2.20}$$

To calculate the probability of the context word given the input word w_t, the Softmax function is applied to the output layer u in the manner shown by Mikolov et al. (2013b, p. 3). This calculates a normalized probability for every context word given w_t. To formalize this in terms of a loss function, the training target shown in equation 2.20 can be rewritten as follows for every input word w_t:

$$\begin{aligned}\mathcal{L} &= -\log p(w_{t-c},\ldots,w_{t-1},w_{t+1},\ldots,w_{t+c} \mid w_t)\\ &= -\log\prod_{c=1}^{C}\frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{T}\exp(u_{j'})}\\ &= -\sum_{c=1}^{C}u_{j_c^*} + C\cdot\log\sum_{j'=1}^{T}\exp(u_{j'})\end{aligned} \tag{2.21}$$

The notation was slightly adjusted, where $u_{j_c^*}$ refers to the index of the output vector u which corresponds to the actual context word c given the input word w_t. In turn, $\sum_{j'=1}^{T}\exp(u_{j'})$ is a summation over all exponentiated output representations $u_{j'}$ of all T number of words given the input word w_t. The calculated loss is then used to update the trainable model parameters Φ and Ψ using gradient descent via a backward propagation function analogues to what is used for standard neural networks. The desired output of the Skip-Gram model is the weight matrix Φ which once the model is sufficiently trained, corresponds to the vector representation or embeddings of the input words.

The same principle shown in the Skip-Gram model can be applied to graphs in a modified version. First, nodes in graph can be one-hot encoded the same way as words. This means, that nodes can be used as input data in a similar fashion as words. Based on this idea, DeepWalk by Perozzi et al. (2014) achieved a big breakthrough for graph representation learning. The DeepWalk algorithm builds on top of the Skip-Gram model and uses fixed-length random walks for learning the node vector embeddings. To provide a better overview of the DeepWalk algorithm,

the pseudo-code is presented in algorithm 1 & 2 (Perozzi et al. 2014, p. 704).

Algorithm 1: DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$
Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ *to* γ **do**
    $\mathcal{O}$ = Shuffle(V)
    foreach $v_i \in \mathcal{O}$ **do**
        $\mathcal{W}_{vi} = RandomWalk(G, v_i, t)$
        SkipGram($\Phi, \mathcal{W}_{vi}, w$)
    end
end

Algorithm 2: SkipGram($\Phi, \mathcal{W}_{vi}, w$)

foreach $v_j \in \mathcal{W}_{vi}$ **do**
    foreach $u_k \in \mathcal{W}_{vi}[j - w : j + w]$ **do**
        $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
        $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
    end
end

The DeepWalk algorithm shows, that for every node $v \in G$ a fixed length random walk is created. Every node on the random walk is used as a one-hot encoded input for the Skip-Gram model. The context nodes of the input node correspond to the input nodes' neighbors on the random walk within the window size w. This procedure is repeated for γ number of random walks which in turn concludes one training epoch. The rows of Φ then correspond to the node embeddings where $\Phi_u = z_u^T$. Lastly, the dot product of any two node embedding vectors, $z_u^T z_v$, approximately equals the probability, that the two nodes co-occur on a random walk as outlined in equation 2.19. Please note, that the DeepWalk algorithm often uses more efficient approximation methods to calculate the loss function shown in equation 2.21. These approximation methods include hierarchical Softmax which makes use of a binary tree or negative sampling. Both approximation methods are outlined in the paper by Mikolov et al. (2013b).

This is in principle the model which will be used to find the node embeddings of a graph. In the application, the Node2Vec algorithm by Grover & Leskovec (2016) will be employed, which is a generalization of the DeepWalk algorithm. Node2Vec allows for the deployment of biased random walks. In particular, it allows to set probabilities as to whether the random walk is biased towards breadth-first search (BFS) or depth-first search (DFS). Depending on the network structure, setting

an appropriate bias can greatly improve the quality of the embeddings. If no bias towards BFS or DFS is set, an unbiased random walk is employed which is when the output of the Node2Vec algorithm corresponds to the output of the DeepWalk algorithm. More precisely, this occurs when the search bias is set to $\alpha = 1$ with $p = q = 1$ as outlined in the Node2Vec paper (Grover & Leskovec 2016, p. 860). The results revealed, that an unbiased random walk embedded the nodes very well. For that reason, the Node2Vec algorithm is not explained in further detail as the relevant parts are covered by the simpler and reader friendlier DeepWalk algorithm. If interested, the pseudo-code for the Node2Vec algorithm is provided in the article by Grover & Leskovec (2016, p. 859).

The resulting node vector embeddings can then be used for downstream machine learning tasks. An additional benefit of graph representation learning is that the nodes can be encoded into an arbitrary number of dimensions. In this sense, graph representation learning can be used as a powerful dimensionality reduction strategy. Further, the node embedding vectors correspond to the features used in the downstream machine learning tasks. The values of the features were learned automatically using the DeepWalk or Node2Vec algorithm. This approach directly takes care of the otherwise tedious feature selection process. With this approach, only the number of features needs to be defined in terms of feature selection. This is a large advantage and can save a lot of time when working with graphs.

2.2.2 Graph Neural Networks

This section provides an overview of the theory for Graph Neural Networks (GNN). Within the family of GNNs, there are a myriad of different methods available and every few months new methods are being published. GNNs currently enjoy a large popularity and benefit from a large research output. This thesis will focus on two popular and established GNN approaches which are:

1. Graph Convolutional Networks
2. GraphSage

Before presenting the two above mentioned methods, a general overview of the GNN framework is given. First the required setup is defined (Leskovec 2021):

- $G(V, E)$ is a graph with a set of vertices and edge connections
- V is a set of vertices
- A is the adjacency matrix of graph G

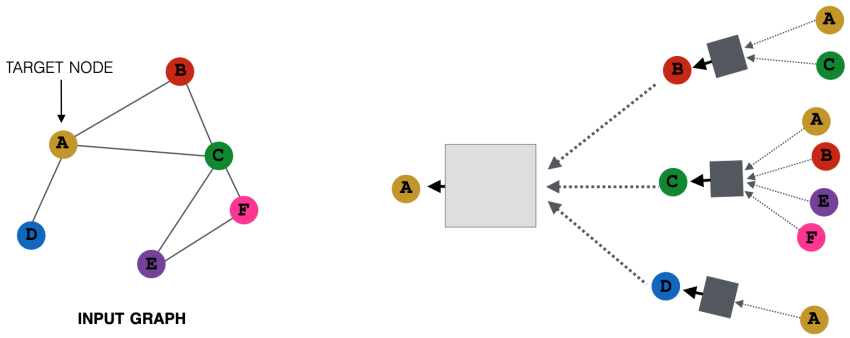

Figure 2.4: GNN Structure
Leskovec (2021)

- $X \in \mathbb{R}^{|V| \times F}$ is a matrix containing the node features
- v is a node $\in V$ and $\mathcal{N}(v)$ is the set of neighbors of v

If there are no node features present, X can be defined as a one-hot encoded vector. A naïve approach for building a GNN would be to append the columns of the adjacency matrix to the feature matrix. This combined matrix would then be used as the input for a standard artificial neural network. The problem with this approach is, that the input is not order invariant and that the trained model cannot be applied to graphs of different sizes (Leskovec 2021).

Modern GNNs have overcome this problem by drawing inspiration from Convolutional Neural Networks (CNN) and its famous filtering mechanism as outlined by Krizhevsky et al. (2012). CNNs typically work with grid structured input data such as pixels of images. The convolutional filter then samples the input grid using a filter with a specified size (e.g. 3×3 grid filter). Similarly GNN sample a graph using the node neighbors $\mathcal{N}(v)$ of node v as a filter. The filter can then be fine tuned in the sense of how many k-hops of neighbors to consider (e.g. 1-hop: immediate neighbors of v, 2-hop: include neighbors of v's neighbors etc.). In terms of implementation, the number of k-hops is set by the number of graph convolutional layers included in the GNN model. A graphical illustration of this mechanism is shown in figure 2.4.

The GNN structure outlined in figure 2.4 shows an example of a 2-hop or 2 layer GNN. The 1-hop convolutional layer considers the neighboring nodes of the target node A. The 2-hop layer considers the neighbors of node A's neighbors. Note, that the target node A is included as an input node in the 2-hop layer. This is reasonable as node A itself is also a neighboring node to its neighbors. Taking the example shown in figure 2.4, the challenge for the GNN is to find node embeddings based

on local network neighborhoods (Leskovec 2021). The node embeddings at layer 0 correspond to the features of the input nodes where $X = H^{(0)}$. A typical procedure for a GNN model is outlined in algorithm 3 (Hamilton et al. 2017, Leskovec 2021, You et al. 2020).

Algorithm 3: Typical GNN algorithm for model training

Input: Graph $G(V, E)$;
input features $\{x_v, \forall v \in V\}$;
node labels $\{y_v, \forall v \in V\}$;
depth/layers K
Trainable and layer specific parameters Θ^k where $W^k \in \Theta^k, \forall k \in \{1, \dots, K\}$;
non-linearity σ;
differentiable aggregator functions $AGGREGATE_k, \forall k \in \{1, \dots, K\}$;
neighborhood function $\mathcal{N}_k : v \rightarrow 2^V, \forall k \in \{1, \dots, K\}$;
loss function $\mathcal{L}$ such as cross entropy CE;
learning rate α

Output: Vector representations $z_v, \forall v \in V$

Initialize parameters Θ from $\mathcal{U}$;
$h_v^0 = x_v, \forall v \in V$
for *Number of epochs* **do**
  for $k = 1 \dots K$ **do**
    forall $v \in V$ **do**
      $h^k_{\mathcal{N}(v)} \leftarrow AGGREGATE_k \left(h_u^{k-1}, \forall u \in \mathcal{N}_k(v) \right)$;
      $h_v^k \leftarrow \sigma \left(W^k \cdot CONCAT(h_v^{k-1}, h^k_{\mathcal{N}(v)}) \right)$;
    end
    $h_v^k \leftarrow h_v^k / \|h_v^k\|_2$
  end
  $z_v = h_v^K, \forall v \in V$;
  $\mathcal{L}(\Theta) = \sum_{v=1}^{|V|} CE(y_v, z_v)$;
  $\Theta = \Theta - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \Theta}$
end

Algorithm 3 is not meant to be considered as a complete overview and should be rather regarded as an example of a typical GNN structure. In addition, one should split the data into training- and validation sets to ensure a good model fit. GNNs are flexible in that a myriad of modifications can be added to the GNN layers similar to the possibilities of CNNs or ANNs. The defining features of different GNN methods usually involve the selection of different message passing methods and aggregation strategies. An excellent overview regarding the design space for GNNs is provided in the articles by You et al. (2020) and Zhou et al. (2020). Of course there are exceptions and alternative procedures exist. The GNN methods evaluated in this thesis and most successful GNNs however tend to follow a variation of the structure shown in algorithm 3.

In terms of interpretation, the output of the first GNN layers in figure 2.4 (gray boxes) corresponds to the hidden layer representations of the direct neighbors of target node A. The output of the final GNN layer corresponds to the node embedding z_A of the target node A. This should appear familiar when comparing this approach to the graph representation learning method outlined in the previous section. GNNs can indeed be used for unsupervised learning tasks such as finding node

embeddings such that similar nodes are embedded close together. Good examples for this are shown in the papers regarding Graph Convolutional Networks by Kipf & Welling (2016) and GraphSage by Hamilton et al. (2017). As shown in algorithm 3, GNNs can be applied directly to a supervised learning task such as classification. This is where GNNs are especially powerful, and differ to the Graph Representation Learning algorithms outlined in the previous section. GNNs are flexible tools and can be used in various settings such as graph representation learning, clustering, classification and link prediction tasks among others.

Having introduced the general functionality of GNNs, the methods Graph Convolutional Networks and GraphSage are introduced in detail in the following two sections.

2.2.2.1 Graph Convolutional Networks

The Graph Convolutional Network (GCN) was introduced by Kipf & Welling (2016) and makes use of simplified spectral graph convolutions. The author Thomas Kipf (2016) provides excellent explanations on his website which is used as inspiration for presenting the theory[1]. As outlined, GNNs typically differ with regards to the type of message passing and aggregation strategy. GCNs make use of the following forward propagation method (Kipf & Welling 2016, p. 2):

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2.22}$$

The variables in equation 2.22 are defined as follows:

- $H^l \in \mathbb{R}^{N\times D}$ refers to the embedding matrix at layer l where N refers to the number of nodes $\mid V \mid$ and D refers to the number of embedding dimensions. The input embedding matrix is set equal to the feature matrix, $H^{(0)} = X$.
- W^l refers to the trainable and layer specific weight matrix for the linear message passing employed in the GCN model.
- $\tilde{A} = A + I_N$, where A is the adjacency matrix of the input graph G. The identity matrix is added so that self-loops are considered. This is necessary as the target node of every layer is considered in the aggregation process as previously outlined.
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is a diagonal matrix containing the degree distributions of the modified adjacency matrix $\tilde{A}$.
- $\sigma(\cdot)$ refers to an activation function such as ReLu or Softmax.

[1]Website Thomas Kipf: `https://tkipf.github.io/graph-convolutional-networks/`

To provide a better overview, the compact notation shown in equation 2.22 is expanded in the following equation for one GCN layer (Dubois 2019):

$$h_{ij}^{(l)} = \sigma\left(\sum_{(i,j)\in\mathcal{N}(v)} \frac{\tilde{a}_{ik} h_{kj}^{(l-1)}}{\sqrt{\tilde{d}_{k,k}\tilde{d}_{i,i}}} W^{(l)}\right) \tag{2.23}$$

In Equation 2.23 $h_{ij}^{(l)}$ refers to the hidden layer representation of node i at layer l considering the set of neighbors j. $h_{kj}^{(l-1)}$ corresponds to the hidden layer representation of node k at layer $l-1$ which is part of the set of neighbors j. In terms of filtering strategy, $(i,j) \in \mathcal{N}(v)$. This means that both set of nodes i and j are neighbors of the target node v at layers l and $(l-1)$ respectively. Linear message passing is then performed for every neighbor in j at layer $l-1$. The node in the graph G at position $\tilde{a}_{ik}$ of the modified adjacency matrix selects the nodes for which a connection between $h_{ij}^{(l)}$ and $h_{kj}^{(l-1)}$ exists. The embeddings $h_{kj}^{(l-1)}$ of the selected nodes are normalized by the symmetric degree distributions of the previous hidden layer node $\tilde{d}_{k,k}$ and the new hidden layer node $\tilde{d}_{i,i}$. Afterwards the normalized embeddings are message passed by multiplying it with the shared weight matrix $W^{(l)}$. Lastly, the sum of the received messages is taken in terms of aggregation strategy and the resulting aggregate is passed through the activation function to yield $h_{ij}^{(l)}$. Note, that the aggregation strategy involves taking a weighted sum thanks to the symmetric normalization.

The detailed explanations given above show the procedure for one layer of a GCN. Returning now to compact notation, this procedure can be expanded for two or more GCN layers. First, the notation is further simplified by defining $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. An example of a two-layer GCN is then given as follows where Z refers to the embedding or output of the target node (Kipf & Welling 2016, p. 3):

$$Z = f(X,A) = \text{softmax}\left(\hat{A}\ \text{ReLu}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) \tag{2.24}$$

Finally, the model parameters are updated analogues to the procedure outlined in algorithm 3. The main distinctive feature for the GCN is the differing forward propagation procedure.

2.2.2.2 GraphSage

This section introduces GraphSage by Hamilton et al. (2017) which can be thought of as the inductive counterpart of the GCN presented in the previous section. Inductive refers to the capability of not only performing machine learning tasks on the graph used for training but to apply the trained GNN model to new and un-

seen graphs. This is a large leap as GCN for instance can only be used to predict unseen nodes on the graph which was used for training. This is very limiting for the application of GNN in a practical setting. GraphSage achieves this by applying different aggregation strategies and sampling the neighborhood $\mathcal{N}(v)$. Specifically, the GraphSage model only considers a fixed number of uniformly random sampled neighbors from $\mathcal{N}(v)$ with depth K. A graphical example of this procedure is shown in figure 2.5:

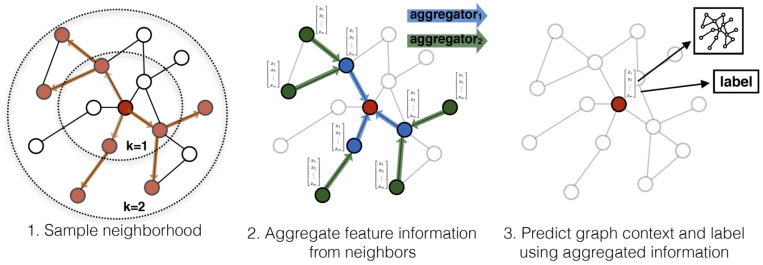

Figure 2.5: GraphSage Sampling
(Hamilton et al. 2017, p. 2)

The steps shown in figure 2.5 are similar to the procedures for GCNs. The pseudo code for the GraphSage forward propagation function is given in algorithm 4 (Hamilton et al. 2017, p. 12).

Algorithm 4: GraphSAGE minibatch forward propagation algorithm

Input: Graph $G(V, E)$;
input features $\{x_v, \forall v \in \mathcal{B}\}$;
depth K;
weight matrices $W^k, \forall k \in \{1, \dots, K\}$;
non-linearity σ;
differentiable aggregator functions $AGGREGATE_k, \forall k \in \{1, \dots, K\}$;
neighborhood sampling functions, $\mathcal{N}_k : v \to 2^v, \forall k \in \{1, \dots, K\}$

Output: Vector representations z_v for all $v \in \mathcal{B}$

$\mathcal{B}^K \in \mathcal{B}$
for $k = K \dots 1$ **do**
  $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^k$
  for $u \in \mathcal{B}^k$ **do**
    $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^{k-1} \cup \mathcal{N}_k(u)$
  end
end
$h_u^0 \leftarrow x_v, \forall v \in \mathcal{B}^0$
for $k = 1 \dots K$ **do**
  for $u \in \mathcal{B}^k$ **do**
    $h^k_{\mathcal{N}(u)} \leftarrow AGGREGATE_k(\{h^{k-1}_{u'}, \forall u' \in \mathcal{N}_k(u)\})$;
    $h^k_u \leftarrow \sigma\left(W^k \cdot CONCAT(h^{k-1}_u, h^k_{\mathcal{N}(u)})\right)$;
    $h^k_u \leftarrow h^k_u / \|h^k_u\|_2$
  end
end
$z_v \leftarrow h^K_u, \forall u \in \mathcal{B}$

Note that algorithm 4 assumes that the parameters of the aggregator functions and

the weight matrices W are known. Algorithm 4 thus shows the forward propagation of a trained GraphSage model. The model is trained by optimizing the model parameters for every batch trained during a specified number of epochs. The training procedure can be implemented by using an adaptation of the general procedure shown in algorithm 3. Note, that the parameters of the aggregation functions and the weight matrices W are all layer specific elements of Θ in algorithm 3. For algorithm 4, $\mathcal{B}$ refers to the mini-batches of vertices taken from the set of vertices V of the graph $G(V, E)$.

There are three aggregator types proposed for GraphSage (Hamilton et al. 2017):

1. Mean aggregation
2. Max pooling
3. LSTM (long short-term memory) aggregation

The three proposed aggregation strategies are briefly introduced as follows:

Mean Aggregation
This type of aggregation is similar to the GCN and takes the average of the received messages from the message passing process. The difference to GCN is that mean aggregation does not rely on the full graph Laplacian and makes use of a slightly different normalization approach. The aggregation process differs to the one shown in algorithm 4 and replaces the procedures in line 9 and 10 with (Hamilton et al. 2017, p. 5):

$$h_v^k \leftarrow \sigma\left(W \cdot \text{MEAN}(\{h_v^{k-1}\} \cup \{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})\right) \tag{2.25}$$

Max Pooling Aggregation
Max Pooling aggregation refers to the application of an element-wise max operator. This means that of the neighbors, only the largest element-wise features are considered. More formally, max pooling aggregation is defined as follows and is used for the aggregation shown in line 9 of algorithm 4 (Hamilton et al. 2017, p. 6):

$$AGGREGATE_k^{pool} = \max\left(\sigma(\{W_{pool} h_{u_i}^k + b), \forall u_i \in \mathcal{N}(v)\}\right) \tag{2.26}$$

Note, that W_{pool} refers to a separate weight matrix for the one layer ANN of the max pooling aggregation. In principle, an arbitrary number of layers could be added for max pooling. The authors Hamilton et al. (2017) however focus on the case with one layer. The parameters of max pooling are learned analogues to the other GraphSage

model parameters.

LSTM Aggregation

LSTM is the last aggregation strategy proposed for GraphSage and uses the LSTM recurrent neural network (RNN) first introduced by Hochreiter & Schmidhuber (1997) as an aggregation strategy. LSTMs are not permutation invariant which is a requirement for the aggregation strategy. The authors propose using a random permutation of the set of node neighbors to counter this problem (Hamilton et al. 2017, p. 5). The LSTM parameters are trained along with the other GraphSage model parameters.

2.3 Graph Generation

This section introduces the Multiplicative Attribute Graph (MAG) model by Kim & Leskovec (2012). This model is used to generate semi-synthetic graphs from feature data as mentioned in the introduction. Originally, the MAG model was introduced for the purpose of generating realistic graphs from feature data and to show that the resulting graph can obey properties of real-world networks. To do so, Kim & Leskovec generated random feature data for which the model parameters were set in such a manner, that the resulting graph adheres to a set of real network properties. These network properties include the emergence of a giant connected component and a power-law or log-normal degree distribution among others. While, the creation of a graph which follows real-world network properties would be desirable, it is not the primary target for this thesis. The main goal of the synthetic graph generation will be to create a graph that provides useful additional information, which can be exploited using graph machine learning. The MAG model is flexible in this sense, that it can create graphs which are not constrained to adhere to a specific set of network properties. Lastly, Kim & Leskovec (2012, p. 138-139) present model parameters which generate graphs that follow real-world network properties. These model parameters can however not be adopted for the task at hand. The parameters were created for a somewhat simpler task, which only involved randomly generated feature data. When working with real feature data, this task becomes more difficult.

The starting point of the MAG model is a matrix of feature data, $X^{N \times F}$, where N refers to the number of observations and F to the number of features. Kim & Leskovec (2012) refer to feature data as attributes and they use the terms interchangeably. For this thesis a distinction is made, where features correspond to the full set of feature data and attributes refer to the features used for generating the

graph, $G(V, E)$. The attribute data, $B^{N\times K} \subseteq X^{N\times F}$, is therefore a subset of the feature data. As a selection criterion, attribute data must be of such a manner for which reasonable link-affinity matrices, Θ_i, can be defined. In addition, attribute data must be discrete for the MAG model. For an attribute variable such as age, this would have to be discretized for use in the MAG model. The link-affinity matrix, Θ_i, is a probability matrix which is used to estimate the probability of two observations, $(a_i(u), a_i(v)) \in B$, to form a connection. As an illustration, a reasonable assumption could be to assume, that people which are of the same age group, are more likely to form a connection/be friends in a social network, compared to people which are not of the same age group. Another example for this could be gender in terms of biological sex which is a classical binary setting for a link-affinity matrix. Examples for binary attribute link-affinity matrices Θ_i are given in figure 2.6.

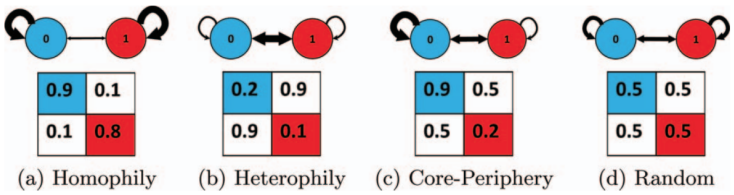

Figure 2.6: Attribute Link-Affinities
(Kim & Leskovec 2012, p. 118)

Figure 2.6 shows 4 types of link-affinity matrices depending on the type of relationship one wants to model. Homophily refers to love of the same which would make a connection between two observations more likely if they have the same attributes. Similarly, Heterophily refers to the love of the different where observations which do not have the same attributes are more likely form a connection. Core-periphery is a special case which can be used to generate realistic social-networks in terms of network properties (Kim & Leskovec 2012, p. 139). As an example, an attribute could indicate whether a person is a member of the local football club. In a core-periphery setting, members of the local football club are very likely to be connected, while non-members have a significantly lower probability of forming a connection. Lastly, random graphs can be generated by setting the link-affinity probabilities to 0.5. Given the type of attributes available in the data sets, graphs will be generated using homophily structures. The attribute link-affinity matrices, Θ_i, are defined for every attribute and can be set for an arbitrary size of categories within an attribute. More formally for each observation $u \in B$ with K categorical attributes of cardinality d_i for $i = 1, 2, \ldots, K$ and corresponding link-affinity matrices $\Theta_i^{d_i\times d_i}$ for $i = 1, 2, \ldots, K$, the probability $P[u, v]$ of a connection between observations (u, v) is defined as (Kim & Leskovec 2012, p. 119):

$$P[u, v] = \prod_{i=1}^{K} \Theta_i \left[a_i(u), a_i(v)\right] \tag{2.27}$$

In equation 2.27, $a_i(u)$ refers to the value of the ith attribute of observation u. A schematic representation of the procedure for a binary link-affinity matrix is shown in figure 2.7.

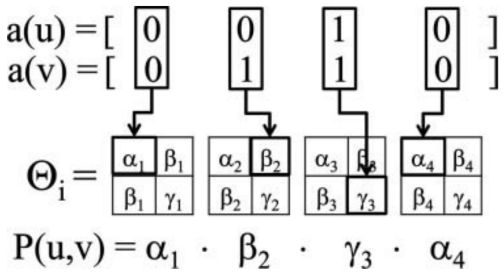

Figure 2.7: Schematic representation of the multiplicative attribute graphs (MAG) model

(Kim & Leskovec 2012, p. 120)

Algorithm 5: Multiplicative Attribute Graph Model

Input: graph node-attribute generation matrix $B^{N\times K}$, where $B \subseteq X^{N\times F}$;
node attribute vector a_i with cardinalities d_i for $i = 1, 2, \ldots, K$;
link affinity matrices $\Theta_i^{d_i \times d_i}$, for $i = 1, 2, \ldots, K$;
Output: adjacency Matrix $A^{N\times N}$ for Graph $G(V, E)$

$B^{K\times N} = B^T$
for $j = 1, 2, \ldots, N$ **do**
  $u = B[:, j]$
  for $k = 1, 2, \ldots, N$ **do**
    $v = B[:, k]$
    for $i = 1, 2, \ldots, K$ **do**
      $P_{j,k} = \prod_{i=1}^{K} \Theta_i[a_i(u), a_i(v)]$
    end
  end
end
$U^{N\times N} = uppertriangular(P)$ with $diag(U) = 0$
for $i = 1, 2, \ldots, N$ **do**
  for $j = 1, 2, \ldots, N$ **do**
    if $U_{i,j} > \mathcal{U}(0, 1)$ **then**
      $\hat{A}_{i,j} = 1$
    end
    else
      $\hat{A}_{i,j} = 0$
    end
  end
end
$A = \hat{A} + \hat{A}^T$

Algorithm 5 generates the adjacency matrix A which is then used for constructing the graph, $G(V, E)$. The observations of the dataset, X, correspond to the nodes present in the graph. More precisely, the order of the generated adjacency matrix, A, corresponds to the ordering of the feature matrix X. Therefore the features can be

assigned to the nodes of the generated graph. The procedure outlined in algorithm 5 can be summarized with following sequential steps:

1. Calculate the connection probabilities P between every observation in the attribute matrix B using equation 2.27.

2. As we only work with undirected graphs, the upper triangular matrix U of P is taken where the $diag(U) = 0$. The diagonal of U is set to 0 to exclude self-loops.

3. For every element in U, draw a random number form a standard uniform distribution $\mathcal{U}(0, 1)$. If the connection probability $P_{u,v} > \mathcal{U}(0, 1)$, a 1 in the preliminary adjacency matrix $\hat{A}_{u,v}$ is recorded. Otherwise a 0 is recorded.

4. The preliminary adjacency matrix $\hat{A}$ is an upper triangular matrix with $diag(\hat{A}) = 0$ and all elements in the lower triangular also being equal to 0. As the target is to create an undirected graph, the corresponding adjacency matrix is symmetric which is why the final adjacency matrix can be created using $A = \hat{A} + \hat{A}^T$.

Chapter 3

Data

This section introduces the datasets used for this thesis. Several approaches and datasets were considered for evaluating the success of graph machine learning on semi-synthetic graphs. In particular, three datasets were considered with varying degrees of success which are:

1. Self launched survey
2. Bank telemarketing dataset
3. US Airline passenger satisfaction survey

The datasets are introduced to the extent that they were successful or useful within the framework of this thesis. In particular, the self launched survey and the bank telemarketing dataset showed to be problematic for different reasons. They however provide valuable insights as to when graph machine learning can be successful. These two datasets are therefore only briefly introduced with the focus lying on providing the insights gained from these "failed" datasets. The detailed introduction and analysis of these datasets is skipped and is deferred to the appendix. The airline passenger satisfaction dataset will be presented in detail, as good results were achieved using graph machine learning. This dataset will also be used for comparing graph machine learning to the standard machine learning methods.

Before introducing the datasets, the programming language and the packages used for the analysis are thankfully referenced in the following section.

3.1 Software

The entire thesis was evaluated using the Python 3.8.10 programming language (Van Rossum & Drake 2009). In addition, following open-source python packages

were thankfully used which are Numpy 1.20.2 (Harris et al. 2020), Matplotlib 3.3.4 (Hunter 2007), NetworkX 2.5.1 (Hagberg et al. 2008), Seaborn 0.11.1 (Waskom 2021), Pandas 1.2.5 (McKinney et al. 2010), Statsmodels 0.12.2 (Seabold & Perktold 2010), Scikit-Learn 0.24.2 (Pedregosa et al. 2011), Tensorflow 2.4.0 (Abadi et al. 2016), Pytorch 1.7.0 (Paszke et al. 2019), deep graph library (dgl) 0.6.1 (Wang et al. 2019), tqdm 4.61.1 (da Costa-Luis et al. 2021) and Node2Vec 0.4.3 (Cohen 2021).

3.2 Self Launched Survey

Initially, the aim was to make use of a self-launched survey which focused on a bank client classification task. The classification task was two-fold in that a simpler task focused on classifying bank clients as to whether they would be interested in investing or not. The second classification task involved classifying clients according to their investment preferences in terms of products (single securities like stocks or bonds, funds, ETFs, unsure). The variables used for the graph creation using the MAG model included mostly demographic data. Additional data was collected by assessing the financial knowledge and behavioral profile of the survey participants by using questions from the financial literacy report of the OECD (OECD 2017). The idea was, that demographic data coupled with the financial literacy questions should be provide a suitable database for the bank client classification task. This hypothesis was based on the professional experience of the author of this thesis having worked for over 10 years as a client adviser for a large Swiss bank.

Unfortunately, only $n = 113$ people participated in the survey which in general is very small for a machine learning task. Further, the graphs generated using the MAG method were not stable. Due to the stochastic element present in the MAG model, the resulting graphs could differ dramatically. This lead to significant performance differences for the different machine learning methods applied to the resulting graphs. Classification accuracies ranged between accuracies of 40 - 95 %. A remedy for this problem could be to assign a fixed probability threshold such as 0.5 in the MAG model. Using a fixed threshold probability, the MAG model would generate deterministic graphs, which are always the same. The downside however is, that this makes the graph generation process less realistic. In a homophily setting this would assume that a connection is formed with any node where $P[u, v] > 0.5$. It is unclear without testing, what impact this would have on the graph generation and whether it is positive or negative. It is well understood, that people often form connections with people that appear unlikely from a probabilistic perspective. This consideration would warrant the generation of stochastic graphs. The main priority is however to create graphs which provide additional information that can be

exploited via graph machine learning. Having this objective in mind, it warrants further investigation for which the results will be presented in section 3.4.2.

The self-launched survey could not be used for any meaningful analysis due to the small sample size. Nevertheless, it provided an interesting follow-up question regarding the question of generating stochastic- versus deterministic graphs. The dataset was discarded for further analysis and the survey data as well as the performed analyses can be found in the appendix.

3.3 Bank Telemarketing Dataset

The bank telemarketing dataset first introduced by Moro et al. (2011, 2014) was considered as a banking related back-up dataset in case that the self made survey did not yield a sufficient number of responses. The bank telemarketing dataset is based on a marketing campaign at a Portuguese bank. The dataset includes demographic data, data regarding the bank client's wealth, contact success during previous campaigns etc. The dataset further provides label data which indicates whether a client invested in a short-term deposit after having been contacted. The dataset is therefore set-up for a binary classification task. The MAG graph generated from the bank telemarketing dataset is shown in figure 3.1.

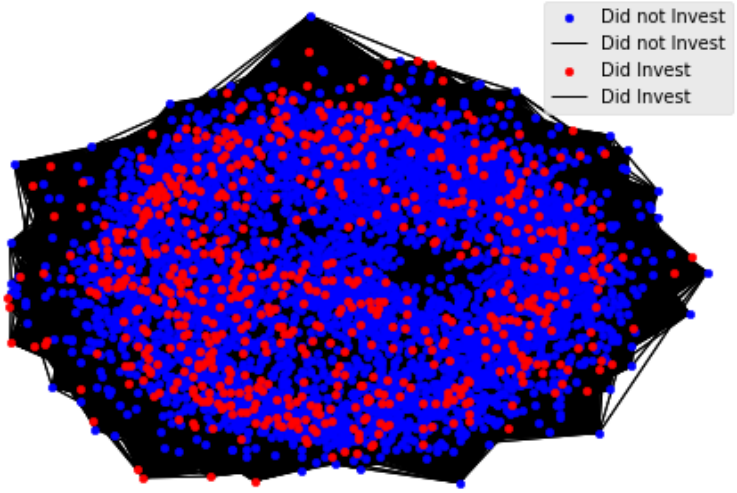

Figure 3.1: MAG graph of bank telemarketing dataset

The red dots in figure 3.1 mark the clients which decided to invest in the short-term deposit and the blue dots did not invest. This figure masks some of the blue nodes due to the figure generation process, the general pattern however is apparent. The red nodes are randomly placed in the network which suggests, that graph machine learning will be of limited use. The graph further shows, that only a relatively

small number of clients appear to have invested in the short-term deposit. To be more precise, only approximately 12% of bank clients invested in the short-term deposit. The dataset is unbalanced which makes the classification task difficult. Graph representation learning using Node2Vec did not provide any useful results and the GNNs also performed rather poorly. In particular, GNNs tended to classify most clients as non-investors and struggled to accurately classify clients which did invest. Due to the unbalanced data, it is loss optimizing for the GNN to predict most nodes as non-investors rather than learning the true label. Table 3.1 shows the confusion matrix of the classification results for the validation dataset (20%) using a GraphSage GNN.

Predicted / **Label**	**Did not invest**	**Invested**
Did not invest	1'026	30
Invested	119	33

Table 3.1: Confusion Matrix Validation Bank Telemarketing Data

The resulting confusion matrix corresponds to an accuracy of approximately 87.67%. The MAG generation process was repeated multiple times for which the GraphSage accuracies ranged between 87 - 90%. Similar results were observed for both graph based methods and standard machine learning methods such as ANNs or support vector machines (SVM).

Unbalanced datasets are part of a larger and common problem in machine learning. Possible remedies might include using loss functions which penalize false classifications harsher than the standard cross entropy loss function used for the GNNs. Alternatively, one could also reduce the data set by dropping observations such that the remaining dataset is balanced. This approach has its own problems as dropping a large number of observations discards a lot of potentially valuable information. It could also put in question the external validity of the model. These comments point to a separate field of research and could be interesting for a future project. These approaches were not researched in detail and should be taken as suggestions.

The failure using graph machine learning methods for this dataset reveals, that GNNs are not an easy remedy for unbalanced data. Perhaps if the network structure provided clusters which corresponded to the labels, GNNs could provide superior results. Given the variables available in the dataset and the limitations of using the MAG method, this was not possible. In order to check, whether network structure could indeed remedy the unbalanced label problem, the label of the bank telemarketing dataset was used for the MAG network generation process. The label

is normally not included as an attribute for the MAG model, as the label data is usually unknown outside of the training dataset. The link-affinity probabilities for the label was set as follows:

$$\Theta_{label} = \begin{pmatrix} 0.95 & 0.25 \\ 0.25 & 0.95 \end{pmatrix}$$

The resulting MAG graph when considering the label is shown in figure 3.2.

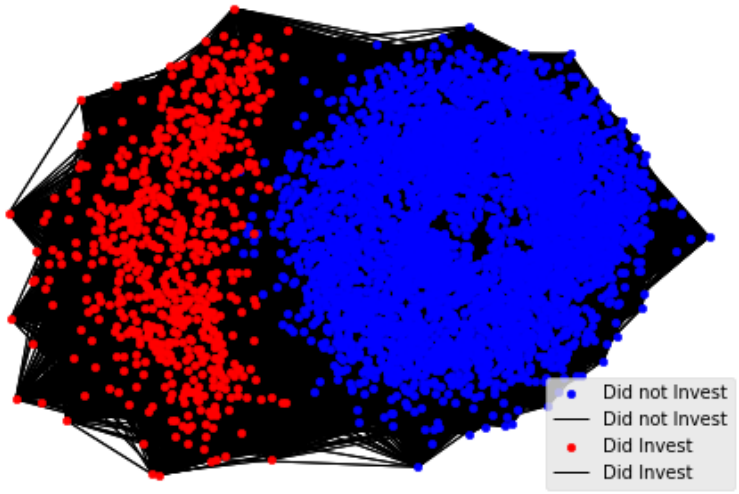

Figure 3.2: Biased MAG graph of bank telemarketing dataset

The nodes shown in figure 3.2 are now nicely clustered according to their label. The GNN method GraphSage achieved an accuracy of over 95% for this graph using otherwise identical feature data and model specifications as before. This is of course a form of cheating, as we cannot assume to know the labels of the graph outside of the training setting. In a real-world application, the model would be trained using label data and then applied to new data which does not contain the label. We would however require the label for the graph generation procedure which makes this not a viable approach. Nevertheless, this result shows extremely well when graph machine learning can yield superior results compared to standard machine learning methods. The key lies in generating a graph with a network structure that corresponds to the label. The network structure need not necessarily be as clearly separated as shown in figure 3.2. A graph which contains local neighborhood clusters which correspond to the same label could yield similar good results. In such a setting, one would have to be careful when defining the number of layers, K, and neighborhood sampling function $\mathcal{N}_k$. A network structure which corresponds to the label could perhaps also be generated without the label to an extent. This would require the attribute

data used in the MAG model to be related with the label. The attributes would have to be substitutes for generating a network structure which correspond to the label. Given the available attributes for the bank telemarketing dataset, this is a difficult task. All features in the dataset have very low correlations with the label. The largest correlation, which is a strong outlier, with the label is call duration with a correlation coefficient ≈ 0.3. This value is rather small and a simulation showed, that it could not be used as a single substitute for the label. Selecting the appropriate attributes and defining the link-affinity probabilities is not a trivial task and requires a lot of trial and error. Unfortunately, for this dataset no appropriate attributes and link-affinities were found that yielded the desired result. For that reason, this dataset was also discarded. The details as well as the analysis can be found in the appendix.

3.4 Airline Passenger Satisfaction Survey

The US airline passenger satisfaction survey was a survey conducted in 2015 by J.D. Power and the dataset was retrieved on the website Kaggle (Power 2015, KAGGLE Inc. 2020). This dataset revealed to be well suited for applying graph machine learning tasks via the MAG method. It further showed to be a competitive dataset for classic machine learning methods. This makes this dataset a suitable candidate for a fair comparison of graph machine learning methods vs standard machine learning strategies. This dataset will be presented in detail as it will be used for the results shown in chapter 4.

An overview of the US Airline Passenger dataset is shown in table 3.2. The correlation heatmap of the dataset is further shown in figure 3.3. The correlation heatmap revealed, that the variables "Departure Delay in Minutes" and "Arrival Delay in Minutes" are highly correlated. As "Arrival Delay in Minutes" has some missing observations, this variable is dropped in favor of "Departure Delay in Minutes". The heatmap and its corresponding correlation matrix reveal, that "Gender" is approximately uncorrelated with any of the other variables. Further, "Departure Delay in Minutes" appears to be approximately uncorrelated with any of the other variables. For that reason, it was tested whether both variables could be excluded. The results however revealed, that the machine learning models performed better if these variables were included in the model. The data shown in table 3.2 and figure 3.3 corresponds to a random sample of 6'000 observations from the training dataset consisting of 103'904 observations. The training graph will be created using this sample of 6'000 observations due to computational time considerations.

Variable	**Description**	**Mean**	**Range**
Gender	Gender of the passengers (Male:0, Female:1)	0.5076	0 - 1
Customer Type	The customer type (loyal customer:0, disloyal customer:1)	0.18	0 - 1
Age	The actual age of the passengers	39.101	7 - 85
Type of Travel	Purpose of the flight of the passengers (Personal Travel:0, Business Travel:1)	0.6891	0 - 1
Flight Distance	The flight distance of this journey	1'197.438	67 - 4'963
Departure Delay in Minutes	Minutes delayed when departure	14.808	0 - 595
Arrival Delay in Minutes	Minutes delayed when arrival	15.159	0 - 589
Satisfied	Satisfaction: Airline satisfaction level(Satisfied, Neutral or Dissatisfaction)	0.4295	0 - 1
Class	Travel class in the plane of the passengers (Eco:1, Eco Plus:2, Business:3)	-	1 - 3
Inflight WiFi service	Satisfaction level of the inflight WiFi service (0:Not Applicable;1-5)	-	0 - 5
Ease of Online booking	Satisfaction level of online booking	-	0 - 5
Gate location	Satisfaction level of gate location	-	0 - 5
Food and drink	Satisfaction level of Food and drink	-	0 - 5
Online boarding	Satisfaction level of online boarding	-	0 - 5
Seat comfort	Satisfaction level of seat comfort	-	0 - 5
Inflight entertainment	Satisfaction level of inflight entertainment	-	0 - 5
On-board service	Satisfaction level of on-board service	-	0 - 5
Leg room service	Satisfaction level of leg room service	-	0 - 5
Baggage handling	Satisfaction level of baggage handling	-	0 - 5
Check-in service	Satisfaction level of check-in service	-	0 - 5
Inflight service	Satisfaction level of inflight service	-	0 - 5
Cleanliness	Satisfaction level of cleanliness	-	0 - 5

Table 3.2: Airline Dataset overview

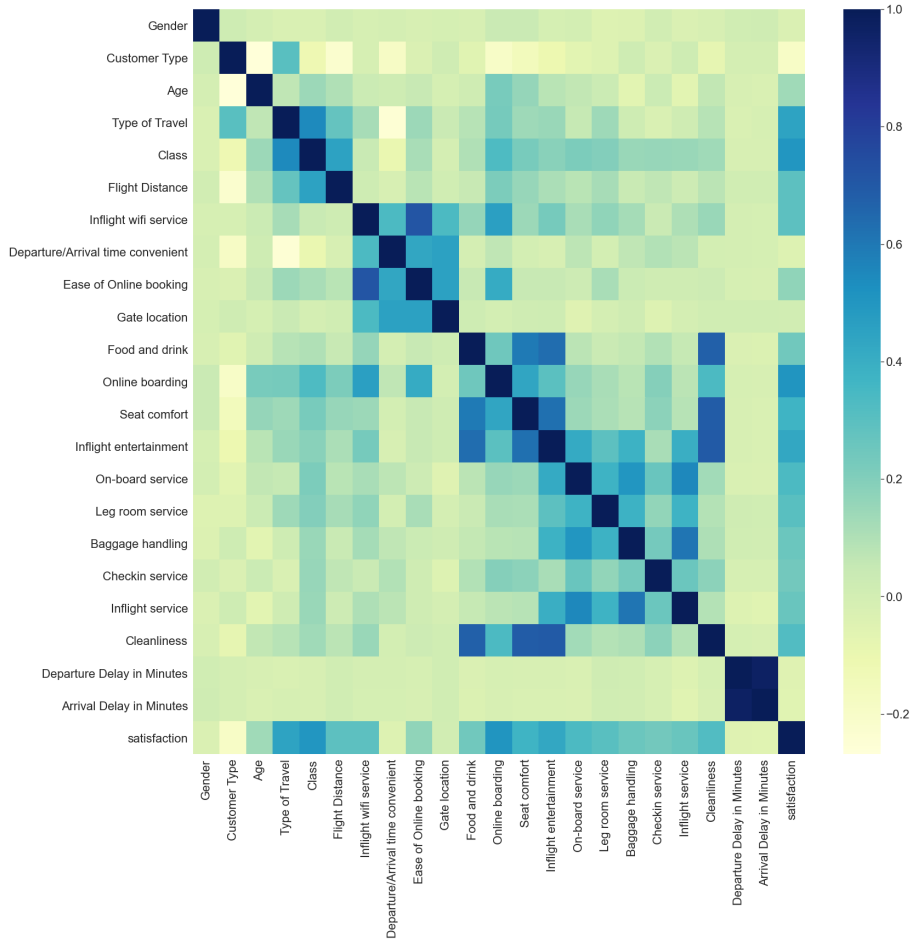

Figure 3.3: Correlation Heatmap of US Airline Passenger Dataset

The variables of the dataset are classified as follows:

- **Categorical Variables:** Gender, Customer Type, Type of Travel, Satisfaction
- **Ordinal Variables:** Class, Inflight WiFi Service, Ease of Online Booking, Gate Location, Food and Drink, Online Boarding, Seat Comfort, Inflight Entertainment, On-Board Service, Leg Room Service, Baggage Handling, Check-in Service, Inflight Service, Cleanliness
- **Numerical Variables:** Age, Flight Distance, Departure Delay in Minutes

The categorical variables were dummy coded and the ordinal variable Class was recorded as shown in table 3.2. The remaining ordinal variables which measure different satisfaction levels were recorded using a Likert scale ranging from $1-5$. Many passengers did not answer all satisfaction level questions. These responses were recorded with a 0. Therefore, the range of values for the satisfaction level variables range from $0-5$. This encoding works well, as a 0 input in a linear pass function of a (graph) neural network will result in a 0 output value. This type

of encoding allows us to deal with missing values for neural networks and other machine learning methods. Lastly, the numerical variables had to be normalized. The popular approach of standardizing the entire dataset was unfortunately not possible, because it had to be ensured, that a 0 was referred to as a missing response. Note, that a 0 for the numerical variables does not correspond to a missing value. The normalizing function employed is defined as follows:

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)} \tag{3.1}$$

In equation 3.1, x refers to the unnormalized variable and x' refers to resulting normalized variable. a defines the lower bound of the normalization range and b is the upper bound. Now, all variables are within a similar range and different scaling should no longer lead to a biasing behavior.

In the following section, the graph generation process for the US Airline Passenger dataset is described in detail.

3.4.1 Graph Generation

To create a graph from the US Airline Passenger Dataset, appropriate attributes must be selected for the MAG model. The selected attributes must be of the type such that realistic probabilities can be assigned. As an example, it is difficult to assign link-affinity probabilities for people who gave ratings regarding the "inflight wifi service". In this case one could assign a probability that people who gave high ratings are more similar with relative ease. However, does this then also translate to people not liking the wifi-service being similar as well? Further, how do we assign probabilities for people who are dissimilar? These considerations make the selection of appropriate attributes difficult. It is therefore important to select attributes for which realistic variables for all of the following three settings can be assigned:

- **Positive similar observations** (e.g. both observations like the service)
- **Negative similar observations** (e.g. both observations dislike the service)
- **Dissimilar observations** (Symmetric for undirected graphs, can be asymmetric for directed graphs)

The attributes were selected using the above mentioned considerations. The selected attributes with the corresponding link-affinity probabilities are shown in table 3.3. The probabilities in table 3.3 correspond to the rows of the link-affinity matrices up to the semi-colon. To give a better overview, the link-affinity matrix for age is shown explicitly as follows:

Variable Name	**Link-Affinity Probabilities**
Gender	0.6, 0.4; 0.4, 0.6
Customer Type	0.8, 0.5; 0.5, 0.8
Age	0.90, 0.80, 0.60, 0,40; 0.80, 0.90, 0.80, 0.60; 0.60, 0.80, 0.90, 0.80; 0.40, 0.60, 0.80, 0.90
Type of Travel	0.80, 0.20; 0.20, 0.80
Class	0.85, 0.60, 0.45; 0.60, 0.85, 0.60; 0.45, 0.60, 0.85

Table 3.3: Link-Affinity Matrices

$$\Theta_{Age} = \begin{pmatrix} 0.90 & 0.80 & 0.60 & 0.40 \\ 0.80 & 0.90 & 0.80 & 0.60 \\ 0.60 & 0.80 & 0.90 & 0.80 \\ 0.40 & 0.60 & 0.80 & 0.90 \end{pmatrix}$$

Age was not only selected as an example as it is the largest link-affinity matrix, it also required some additional data transformation. The MAG model can only consider discrete variables which is why age had to be binned into discrete categories. Age was binned into 4 bins with 0 if age < 26, 1 if $26 \leqslant$ age < 39, 2 if $39 \leqslant$ age < 50 and 3 if age $\geqslant 50$. These bins were chosen according to the interquartile lengths present in the distribution of the variable age.

For assigning link-affinity probabilities there exist no clear rules as to how these are to be defined. The article by Kim & Leskovec (2012, p. 118) shows the 4 common structures of homophily, heterophily, core-periphery and random for creating graphs. Given the selected attribute data, the homophily setting was most appropriate for all link-affinity matrices. In this setting, observations which are similar have a higher probability of forming a connection compared to dissimilar observations. The probabilities were assigned based on personal intuition and trial and error. Several graphs were created using different probabilities, where the probabilities shown in table 3.3 generated the most reasonable graphs. There is however no exact science or selection criteria which can be applied for selecting attributes and defining the link-affinity probabilities.

As mentioned in the previous section, a sub-sample of 6'000 observations was retrieved from the training dataset consisting of 103'904 observations. With this random sub-sample and the attributes shown in table 3.3, the adjacency matrix for the resulting graph, $G(V, E)$ was generated using algorithm 5. The random sub-sample was retrieved due to the computational cost involved for applying algorithm 5. Simulations involving creating graphs with different random sub-samples suggest, that the random sub-samples are representative for the entire dataset. This suggestion

is supported when comparing the summary statistics of the sub-sample with the full-dataset. The generated graph is shown in figure 3.4.

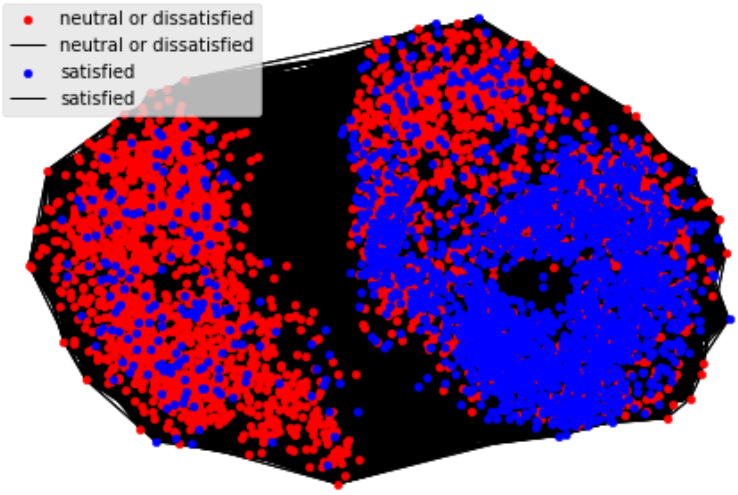

Figure 3.4: Graph of US Airline Passenger Dataset

The network shown in figure 3.4 show the emergence of two primary clusters. In addition one can see, that most satisfied airline passengers appear to be grouped together in the right cluster. To gain a deeper understanding of the dynamics involved in the network formation, the nodes of the network are plotted excluding edges in figure 3.5.
Figure 3.5 plots the nodes of the network for the label "Satisfaction" and the 5 attributes used for the graph generation. The transparency of the plot was set to $\alpha = 0.6$ to avoid covering nodes during the graph plotting process. Figure 3.5 reveals interesting associations. First it is shown, that a larger number of people traveling for business purposes are satisfied compared to people traveling for personal reasons. This association becomes clear when comparing the satisfaction plot with the type of travel plot. In relative terms, only approximately 9.8% of passengers traveling for personal reasons were satisfied compared to business travelers with a 57.8% satisfaction rate. Interestingly, passengers traveling for business purposes hold somewhat expected characteristics such as:

1. Most business class passengers are satisfied.

2. Older passengers appear to be more satisfied which is largely associated with booking more business class tickets. The age plot however reveals, that older passengers tend to be mostly satisfied even when booking economy class.

3. Most loyal customers book business class. There is some overlap where loyal customers book economy class tickets. The reverse is true as well, where a cluster of disloyal customers book business class.

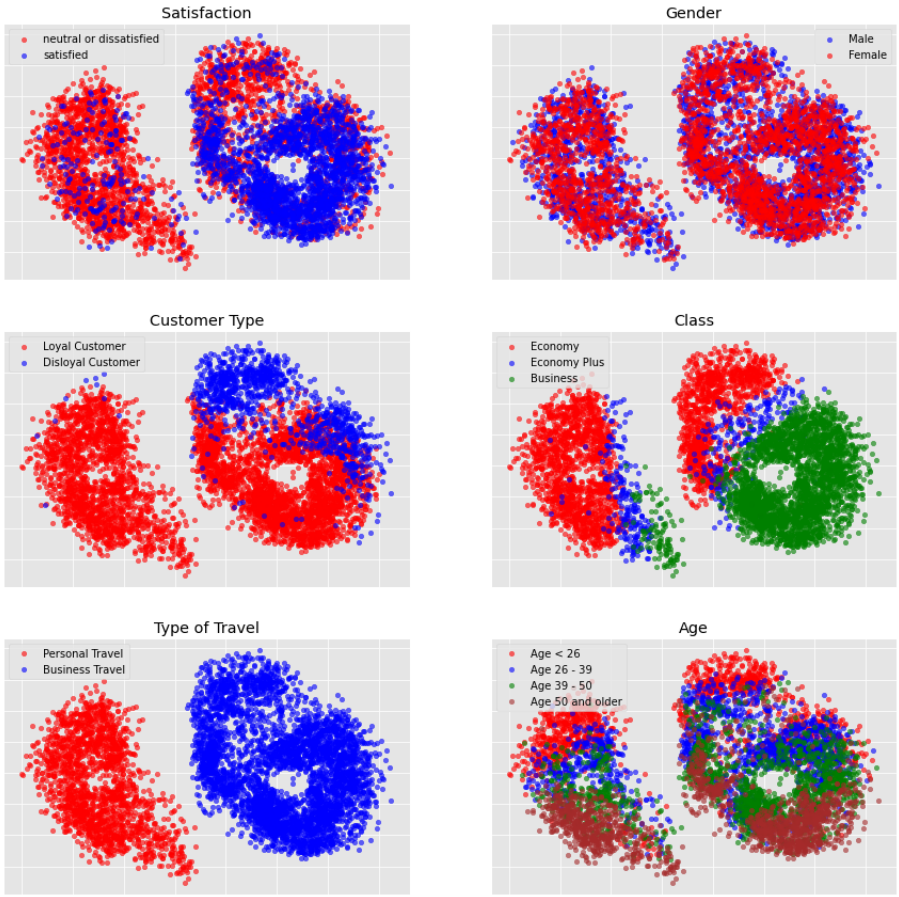

Figure 3.5: Graph Nodes of US Airline Passenger Dataset

Passengers traveling for personal reasons do not appear to adhere to the characteristics or associations shown for business travelers. The only distinctive character is, that almost all passenger traveling for personal reasons are loyal customers. This fact does however not appear to be associated with satisfaction. This appears to be a rather bizarre finding. Upon further reflection, this could point to a sampling bias of passengers traveling for personal reasons based on following considerations:

1. It makes sense, that almost only loyal passengers participated, as when traveling, most people do not participate in surveys. This is especially true for disloyal customers. Business travelers for comparison might give feedback due to company policies.

2. It is common, that dissatisfied people are more likely to give feedback, while satisfied passengers are less likely to participate in the survey. Again, the data regarding business travelers might be more reliable here due to company mandated survey participation.

3. Perhaps business travelers fly more frequently than passengers traveling for personal reasons. This could incentivize frequent business travelers to give

feedback as they would benefit most of service improvements. Infrequent personal travelers might be less incentivized, as they are less affected by service improvements.

Last but not least, gender does not appear to form any distinguishable clusters. For that reason, it was considered to omit this attribute for the graph generation process. This was tested and the resulting graph yielded a similar graph. The graph was however more spread out and the neighborhood structures were less clear. In addition, the graph without gender did not perform as well in the subsequent machine learning tasks. For those reasons, gender was kept for the graph generation process.

To provide some more context regarding the graph structure, some graph theoretical metrics are provided. The degree distribution as well as the distributions of the centrality measures: eigenvector centrality, closeness centrality and betweenness centrality are shown in figure 3.6.

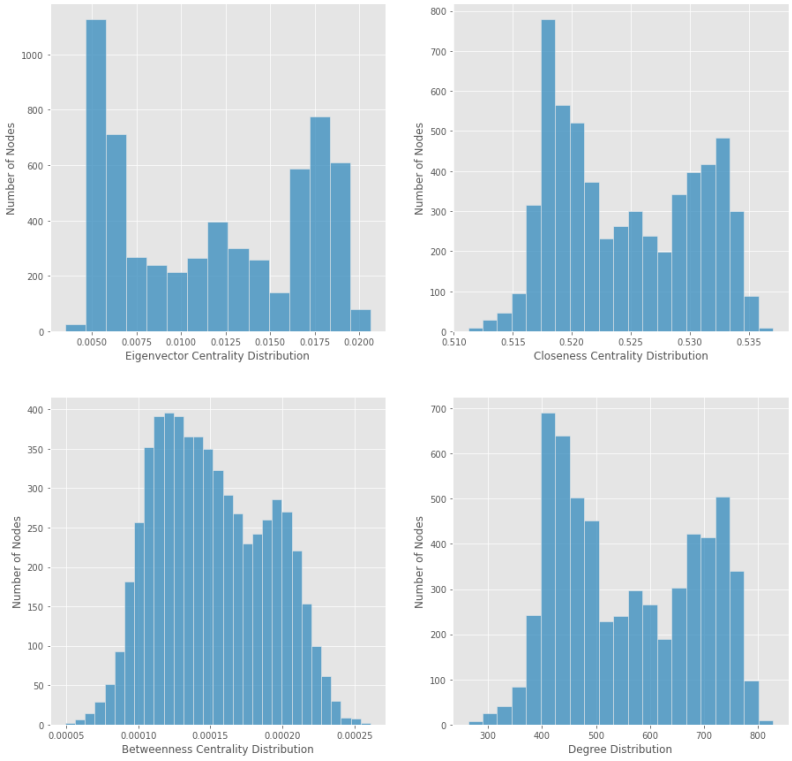

Figure 3.6: Graph Statistics

The network has a density of approximately 0.0933. This means that 9% of the potential number of connections formed in the network. Nevertheless, when looking at the degree distribution histogram we can see, that a all nodes have a large number of connections ranging between 263 - 828 with an average of 559.63 connections. Further, the distribution has two modes which likely correspond to the two main

clusters shown in figure 3.5. The eigenvector centrality distribution shows, that all nodes have a very low centrality measure and therefore none of the nodes appear to have a large impact in terms of eigenvector centrality. The closeness centrality distribution shows, that all nodes have an average closeness centrality ranging from 0.51 - 0.53. This means that every node is similarly connected and has an average impact for disseminating information across the network. Lastly, the betweenness centrality distribution reveals that there are no bottle-necks through which information flows.

As a reference point, it is important to compare the properties of the created graph to real world graphs. In our case, the most appropriate for comparison is a social network. Common structures of social networks are (Watts & Strogatz 1998, Newman et al. 2006, Newman 2010, Kim & Leskovec 2012):

1. Degree distributions often follow a power law distribution
2. Emergence of a giant connected component
3. Core-periphery structure

This power law degree distribution and emergence of a giant connected component creates network structures that have an onion (core-periphery) structure (Kim & Leskovec 2012, p. 121). This indicates, that most social networks have a few very highly connected nodes and many nodes with few connections. This creates a right skewed degree distribution which also lead to a right skewed eigenvector centrality and closeness centrality distribution. For the betweenness centrality, we would also expect, that more central nodes in a core-periphery network structure would exhibit some bottle-neck properties for the central nodes. For that reason we would expect central nodes to have a higher betweenness centrality. When we look at the distributions in figure 3.6 this does not correspond to the power law degree distribution and the effect this has on the centrality measures observed in real social networks. This indicates, that the graph created with the MAG method does not share the properties observed in real social networks. In order to generate a graph which shares the properties of real graphs, one would have to adapt the link-affinity properties to the core-periphery setting as shown in figure 2.6. Forcing this core-periphery structure is however not purposeful for this thesis. Further, the aim of this thesis is not necessarily to create realistic graphs rather than to create useful graphs for graph machine learning. The results in chapter 4 reveal, that the graph is indeed useful for graph machine learning. For that reason, the generated graph shown in this section is kept for further analysis.

3.4.2 Stochastic vs. Deterministic MAG

As addressed in section 3.2, the question was raised as to whether the MAG should form connections between observations stochastically or whether a deterministic threshold probability yields better results. The first insight gained when investigating this question lies in the fact, that the probability of two observations is generally very low. When setting the threshold probability for a connection between two observations u and v to 0.5, not a single connection was made. This makes sense, as the probability for a connection decreases by design as the number of attributes increases. This is implicitly shown in equation 2.27, where the product of probabilities is bound to decrease. For this reason, it is suggested to limit the number of attributes to $K = \rho \log_2 N$ for some constant ρ (Kim & Leskovec 2012, p. 122). The number of attributes were selected accordingly such that $K \leqslant \log_2 N$. The US Airline Passenger dataset was used to generate a deterministic graph. The threshold probability for a connection was set to 0.2 in the MAG model. The MAG yielded several disconnected graphs which were for the most part clustered according to their group memberships. Figure 3.7 shows the graphs for the label and the attributes, where the edges were removed. The subgraphs are unfortunately plotted small, however all subgraphs combined include all 6'000 nodes. The plots are meant to provide a high-level overview of the generated graphs.

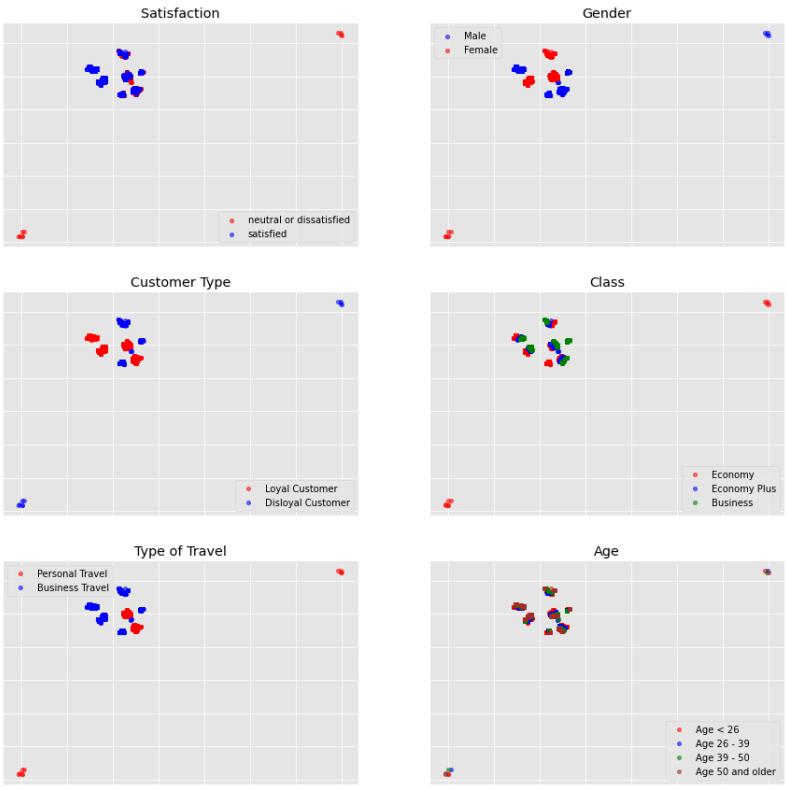

Figure 3.7: Deterministic MAG graph

The deterministic MAG model crates disconnected graphs which form clusters based on node similarities. In terms of performance, the accuracy and loss behavior of both the deterministic- and stochastic generated graphs are virtually identical. This was true for all three datasets considered in this thesis. For the purpose of visualization, stochastic graphs appear to be more useful, as one can identify the different cluster on a single connected graph. For this reason, the stochastic graph generation process is kept. Nevertheless, deterministic graph generation appears to be useful if one wants to separate nodes in to more homogeneous graphs. These graphs could then be used for subsequent machine learning tasks with a cluster specific task in mind. The clusters further could provide information as to which clusters tend to be more or less satisfied. This is an area which could be interesting for future research.

Chapter 4

Results

In this chapter the results for the US Airline Passenger Dataset are presented. First, the model specifications and results of graph representation learning and GNNs are presented. Afterwards, the graph machine learning results are compared to the results using "standard" machine learning methods. For the comparison, the methods logistic regression (Cramer 2002), naive bayes (Zhang 2004), support vector machines (SVM) (Platt et al. 1999, Chang & Lin 2011), Random Forest Classifier (Breiman 2001), AdaBoost Classifier (Freund & Schapire 1997, Hastie et al. 2009), Quadratic Discriminant Analysis (QDA) (Tharwat 2016) and Artificial Neural Networks (McCulloch & Pitts 1943) are considered.

4.1 Graph Representation Learning

The graph generated in section 3.4.1 using the MAG method was used for graph representation learning. The Node2Vec algorithm using an unbiased walk was employed for learning the 2-dimensional node representations of the graph. The Node2Vec algorithm was employed using following model parameters:

- Embedding size d: 2
- Random walk length t: 8
- Number of random walks γ: 100
- Window size w: 10
- Node batch size: 2
- Return parameter $p = 1$
- In-out parameter $q = 1$

With the specified return- and in-out parameters, the Node2Vec output corresponds to the DeepWalk output. The resulting node embeddings were then used as inputs

for standard machine learning methods. The resulting node embeddings are shown in figure 4.1.

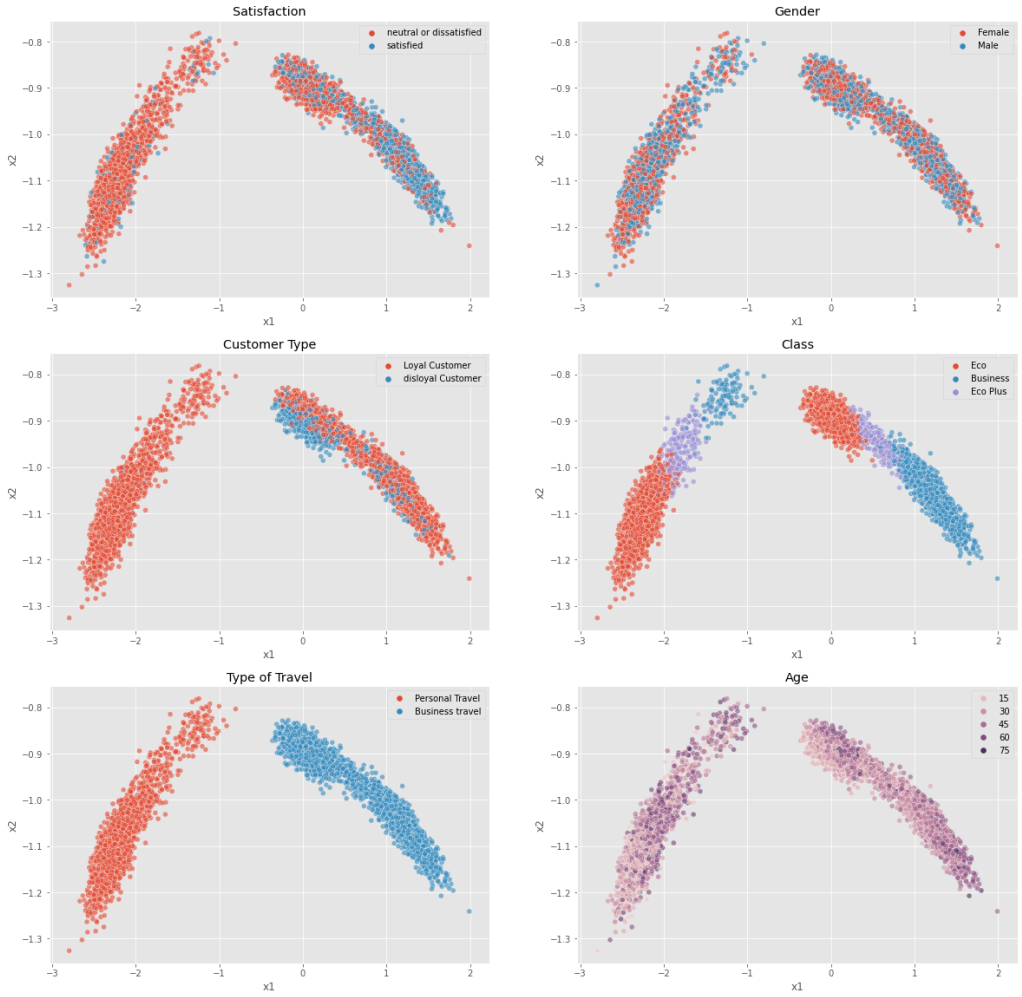

Figure 4.1: Node2Vec embeddings

The plotted node embeddings in figure 4.1 reveal interesting neighborhood structures. First, the nodes are split according to their type of travel. This corresponds to the two main clusters shown in figure 3.4. Secondly, the node embeddings are grouped in a nice and orderly fashion. The node embeddings are part of Euclidean space, which is why the plots shown in figure 4.1 are proper scatter plots. This allows for direct comparison of nodes and the groups which they belong to. The 2-dimensional node embeddings are thus useful for gaining insights via data visualization.

The node embeddings were used as input data for the three standard machine learning methods shown in table 4.1.
The results show, that Node2Vec model was not successful for classifying passengers according to their satisfaction. Applying the trained models to the test data

ML Method	Training Accuracy	Validation Accuracy	Test Accuracy
Logistic Regression	77.04%	76.16%	57.05%
Support Vector Machine	76.71%	77.00%	39.08%
ANN	76.78%	78.08%	29.48%
Random Forest	100%	73.08 %	40.63%
AdaBoost	76.96%	77.00%	58.22%
Naive Bayes	75.73%	76.43%	56.97%
QDA	75.52%	77.33%	57.00%

Table 4.1: Node2Vec Classification Results

yielded even poorer results. When looking at the satisfaction scatter plot in figure 4.1, it becomes obvious why the downstream machine learning tasks were unsuccessful. Node2Vec generated very good node embeddings for the attributes to the extent that clusters exist within the original graph. The label satisfaction could not be used as an attribute for generating the graph, as this would be unrealistic in practice. As the label is not considered for the graph generation process, Node2Vec does not create embeddings which directly consider the label. The label is only considered to the extend that the attributes create structures which are related with the label. For that reason, the success of any downstream machine learning method will be limited to the extent that the node embeddings capture relevant information for predicting the label.

Alternative model specifications were tested, which mainly included learning higher dimensional node embeddings. These node embeddings however did not yield better results. Given the almost identical accuracies, the more parsimonious model with 2 dimensions is preferred. As a final test, the node embeddings were joined to the feature data presented in section 3.4. This data was then used as the input data for the downstream machine learning models. For the training- and validation data, this approach yielded excellent results with accuracies often being close to 95%. For some of the models such as the Random Forest classifier, this also translated into good accuracies for the test data. Unfortunately, the results presented in section 4.3 show, that better accuracies are achieved only using the feature data. For that reason, joining feature data with node embeddings is not a recommended approach for the US Airline Passenger dataset.

4.2 Graph Neural Networks

This section presents the results using graph neural networks. As mentioned in section 2.2.2, GCN by Kipf & Welling (2016) and GraphSage by Hamilton et al. (2017) are used for classifying the satisfaction of the US airline passengers. The model specifications and results are presented in the following sections for both

methods.

4.2.1 Graph Convolutional Network

The GCN was designed using the forward propagation function similar to the function shown in equation 2.24. The only difference is, that the output layer is activated using the logSoftmax function instead of the Softmax function. The outputs of both activation functions theoretically the same. The logSoftmax function is however numerically more stable, as it internally makes use of the log-sum-exp trick[1].

$$Z = f(X, A) = \text{logSoftmax}\left(\hat{A}\ \text{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) \tag{4.1}$$

The node features X include to the 21 explanatory variables of the US airline passenger dataset as described in section 3.4. The categorical variables were one-hot encoded, which is why the feature matrix includes 24 variables. The hidden layer size of the convolutional layers was also set to 24 and the output layer was set to size 2 for the binary classification task. The training loss was calculated using cross-entropy loss and the model parameters were updated using the Adam optimizer (Kingma & Ba 2015) with the learning rate set to 0.002. Essentially, algorithm 3 ca be applied by replacing the forward propagation with equation 4.1 and updating the model parameters using the Adam optimizer instead of standard gradient descent. Different model specification were tested, however the chosen specifications appeared to perform best. The GCN required approximately 1'000 epochs to finish training. This large number of epochs is due to the full-batch training procedure employed for the GCN. The resulting loss- and accuracy plots are shown in figure 4.2.

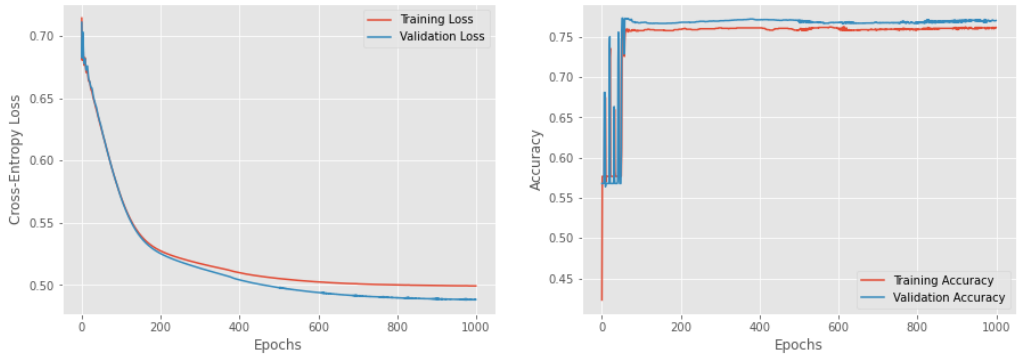

Figure 4.2: GCN Loss- and Accuracy Plots

The resulting confusion matrices and the final accuracies are further presented in tables 4.2 & 4.3.

[1]See for reference: `https://gregorygundersen.com/blog/2020/02/09/log-sum-exp/`

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	793	233
Satisfied	191	562
Accuracy	76.17%	

Table 4.2: Confusion Matrix Training Data

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	1'866	531
Satisfied	440	1'384
Accuracy	77.00%	

Table 4.3: Confusion Matrix Validation Data

Graph convolutional networks are designed to be used in a transductive setting and the model cannot be applied to new and unseen graphs. In this setting 30% of the dataset was used for training and 70% of the data was used for validation. This is done by masking the nodes in the graph which are part of the validations dataset. The masked nodes are considered in the in the neighborhood function $\mathcal{N}(v)$, they are however not considered as target nodes. Therefore, masked nodes are not use for updating the model parameters. The fact, that GCNs cannot be used in an inductive setting is very limiting. Further, the GCN requires over 1'200 epochs to start finish training and yields only mediocre results in terms of accuracy and model fit. A reason for this could be that only a full-batch implementation for GCN was introduced. In addition, GCNs always consider the entire neighborhood set. Sampling the neighborhood and mini-batch training which is an important ingredient of GraphSage and is introduced as an improved and superior GNN method in the following subsection.

4.2.2 GraphSage

For GraphSage the exact same data input was used as for the GCN. The GraphSage model included 2 convolutional layers where both have a hidden layer size of 128 and the output layer is of size 2. The model is defined using an adaptation of algorithm 4. Similar as for the GCN, the training graph is split into 80% training and 20% validation using node masking. The loss is calculated using cross-entropy and the model parameters are updated using the Adam optimizer with the learning rate set to 0.002 and is an adaptation to the model updating procedure shown in algorithm 3. The output of the first convolutional layer is activated using the ReLu function and the output of the second convolutional layer is activated using the log-softmax function. The model was trained using a mini-batch size of 50 nodes and

the neighborhood function $\mathcal{N}(v)$ randomly sampled 10 neighbors 2-hops away from the target node and randomly sampled 5 nodes at a 1-hop distance from the target node. This corresponds to the steps shown in figure 2.5. To improve the robustness of the model, lastly a dropout rate of $p = 0.02$ was set. GraphSage was run using the different aggregation strategies mean, LSTM, max-pooling. In addition, sum-pooling was added as an additional aggregation strategy as suggested by Xu et al. (2018). Sum-pooling aggregation follows the same procedure as max-pooling with the only difference being that element-wise sum-pooling is performed and not max-pooling. This aggregation method has been shown to be more robust for graph neural networks, as it allows the GNN to better distinguish different graph structures. It is important to note, that the Graphsage sum-pooling strategy employed does not correspond to the Graph Isomorphism Network introduced by Xu et al. (2018). This article rather provided the inspiration for applying sum-pooling for Graphsage.

The model for each aggregation strategy was trained using 400 epochs. The training- and validation results using the training graph are presented for every aggregation strategy. In addition, the trained models are applied to a new unseen graph also consisting of 6'000 nodes to check for the inductive capability of the Graphsage model.

Mean Aggregation

In figure 4.3 the training- and validation loss as well as the accuracies for the mean aggregation model is shown.

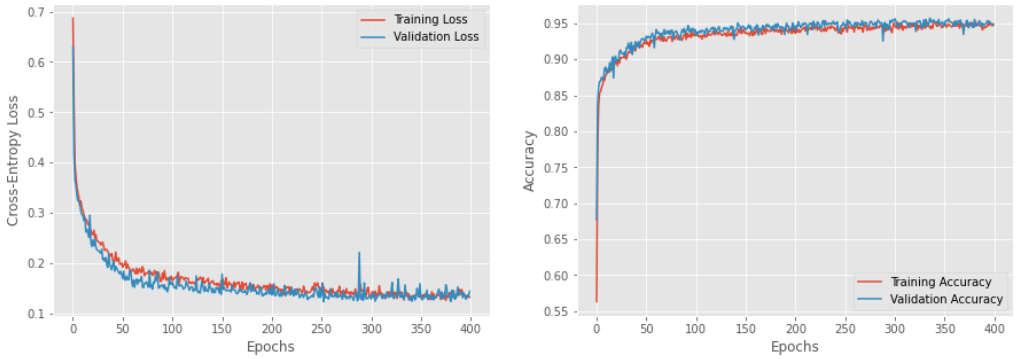

Figure 4.3: Mean Aggregation Loss- and Accuracy Plots

The training accuracy is 94.91% and the validation accuracy is 94.80% after 400 epochs. The model resulted in a test accuracy of 94.08% with the following confusion matrix shown in table 4.4:

Label \ Predicted	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	3'278	88
Satisfied	267	2'367
Accuracy	94.28%	

Table 4.4: Test Confusion Matrix Mean Aggregation

LSTM Aggregation

Figure 4.4 shows the training- and validation loss and accuracy using LSTM aggregation.

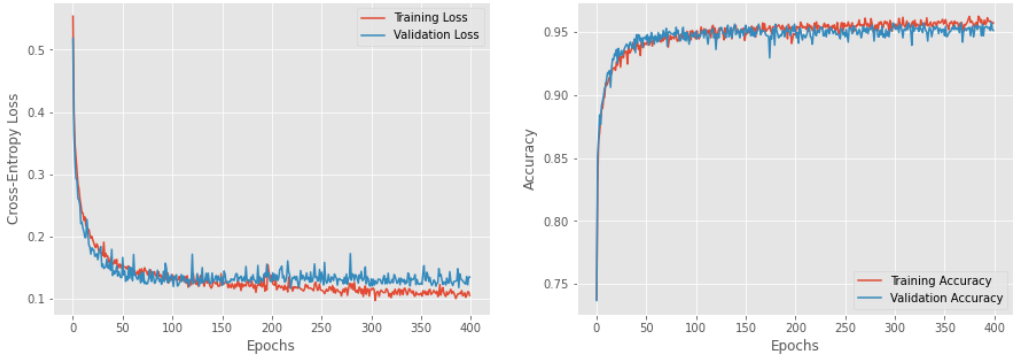

Figure 4.4: LSTM Aggregation Loss- and Accuracy Plots

The training- and validation accuracy after 400 epochs was 95.75% and 95.14% respectively. Here again, we can see that the training behavior is relatively good with a small over-fit which remains stable. The results for the test graph are shown in table 4.5.

Label \ Predicted	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	3'288	78
Satisfied	268	2'366
Accuracy	94.23%	

Table 4.5: Test Confusion Matrix LSTM Aggregation

Sum-Pooling Aggregation

Sum-pooling is shown to provide more consistent results as it prevents the GNN from being confused (Xu et al. 2018). This aggregation method provides solid results as shown in figure 4.5.

The plots show that sum-pooling trains very well. The training accuracy is 94.60% and validation accuracy is 94.97% after the model finished training. The results for the test graph are shown in table 4.6.

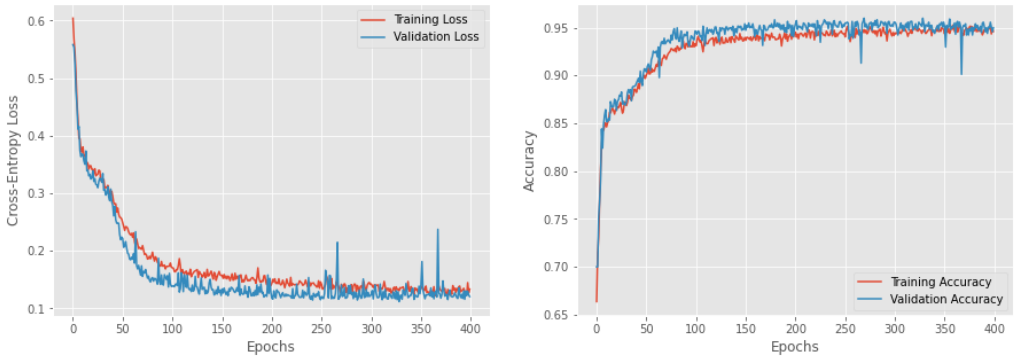

Figure 4.5: Sum-Pooling Aggregation Loss- and Accuracy Plots

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	3'235	131
Satisfied	213	2'421
Accuracy	94.27%	

Table 4.6: Test Confusion Matrix Sum-Pooling

Max-Pooling Aggregation

The last aggregation method is max-pooling which yielded similar results as sum-pooling. The training- and validation loss as well as the accuracies are shown in figure 4.6.

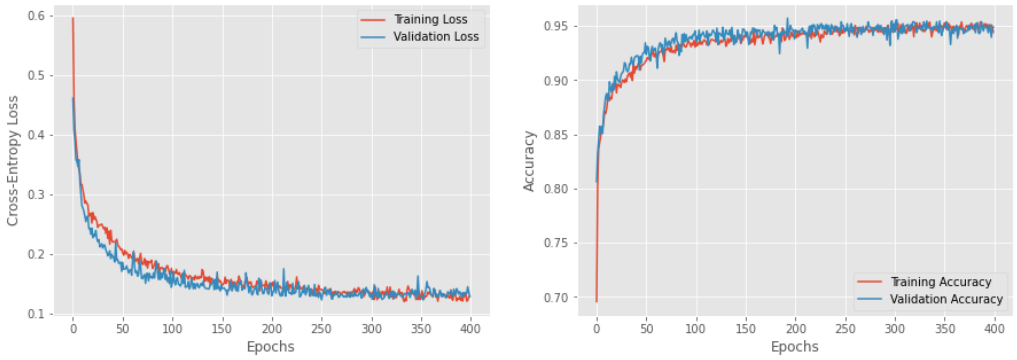

Figure 4.6: Max-Pooling Aggregation Loss- and Accuracy Plots

The training model resulted with a training accuracy of 94.43% and validation accuracy of 94.88%. The results for the test graph are shown in table 4.7.

4.2.3 GraphSage Robustness Simulation

The loss- and accuracy plots reveal mostly a relatively good model fit. LSTM aggregation however shows a moderate over-fit and is indicative of a more general issue shown to be true for all aggregation strategies. Simulations revealed, that for the same graph the trained model would either over-fit, fit perfectly or have a training

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	3'235	131
Satisfied	212	2'422
Accuracy	94.28%	

Table 4.7: Test Confusion Matrix Max-Pooling

pattern in which the validation loss was smaller than the training loss. An analysis revealed, that the culprit for this behavior was the dropout rate of 2% as well as the random assignment of graph nodes into training- and validation data. If the dropout rate was set to 0, the model always tended to over-fit, where the training accuracy would approach 100% and the validation accuracy would stagnate around 93-94%. Unfortunately, when applying this model to an unseen test graph, the accuracy results were somewhat lower at 91-92%. For this reason, a dropout rate at 2% was set to avoid this over-fitting problem which in turn yielded better test results. This has the consequence, that it is more difficult for the model to make predictions for the training data compared to the validation data which uses no dropout rate. This obstacle however has the downside, that it makes the model sensitive to the node assignments during the random train- and validation split. Some nodes are more difficult to learn depending on their features and neighbors. This means, that if the training set includes more difficult nodes on average than the validation set in conjunction with the dropout rate as an additional obstacle for the training set, the validation loss becomes lower than the training loss. The same is true in reverse, which leads to an over-fit. Lastly, the if the nodes in both the training- and validation set are on average of similar difficulty a good model fit is achieved as shown in the loss- and accuracy plots.

Nevertheless, the training- and validation results shown are representative for the model in general. In order to show this, a simulation was run using a training- and validation graph consisting of 6'000 nodes and a test graph also consisting of 6'000 nodes. For the simulation, max-pooling aggregation was selected using the same model specifications as previously. The model was trained during 100 experiments, where for every experiment a new random training- and validation split node assignment was performed. The trained model was then applied to the test graph where the loss- and accuracy was measured. The average results of the 100 experiments are shown in table 4.8 and figure 4.7.
Max-pooling was selected as this corresponds to the recommended aggregation strategy (Hamilton et al. 2017, p. 9). For the remaining aggregation strategies, only 10 experiments were run due to time considerations as 100 experiments require approx-

Metric	Training Set	Validation Set	Test Set
Average Accuracy	94.37%	94.23%	93.21%
Average Cross-Entropy Loss	0.3684	0.3693	0.3792

Table 4.8: Average Simulation Results

imately 15 hours to complete. The results using only 10 experiments yielded similar results, with test accuracies ranging predominantly from 93-94%.

4.3 Result Comparison

The results shown for the three models Node2Vec, GCN and GraphSage are compared to the results using "standard" machine learning methods. This is done to assess to what extent, synthetic graph generation is a useful approach for machine learning. This comparison is an important ingredient for answering the research question. These methods only consider the feature data and do not consider any network connections. These methods are therefore simpler and do not require a graph to be generated. A comparison of the different results is shown in table 4.9.

Method	Training Accuracy	Validation Accuracy	Test Accuracy
Logistic Regression	88.16%	87.08%	86.43%
Naive Bayes	87.46%	85.83%	85.6%
AdaBoost	93.54%	93.08%	92.45%
Random Forest	100%	95.08%	94.38%
SVM	94.42%	93.5%	92.45%
ANN	96.47%	94.67%	93.32%
Node2Vec	77.61%	76.58%	-
GCN	83.73%	82.85%	-
GraphSage (Mean Aggregation)	94.98%	94.59%	93.38%
GraphSage (LSTM Aggregation)	94.44%	94.09%	93.47%
GraphSage (Sum-Pooling)	94.33%	94.2%	94.17%
GraphSage (Max-Pooling)	93.96%	93.84%	93.8%

Table 4.9: Result Comparison

Table 4.9 shows, that GCN and Node2Vec are not competitive when using MAG generated graphs. GraphSage is however a serious competitor and appears to be the second best method overall. The Random Forest classifier is consistently the best method regardless of the training- and validation data as well as the test data used. The GraphSage models are consistently shown to be the second best method. The ANN model performs similarly well in terms of achieving a similar test accuracy. Similar as with the GNNs, it is important to evaluate the model fit to draw a more definitive conclusion which is shown in figure 4.8.

Figure 4.8 shows, that the ANN model has a relatively strong over-fit compared to the GNN models. This observation is consistent for every ANN model trained. In addition, different number of hidden layers, hidden layer sizes and dropout rates

were tested, which all resulted in a clear over-fit or bad training behavior. For that reason, the ANN is deemed to be inferior to the GNN, even if the test accuracies are very similar.

AdaBoost and SVM also are shown to provide very good results, however yield slightly lower test accuracies compared to the previous methods. Lastly, naive bayes and logistic regression clearly yield inferior results.

Figure 4.7: Simulation Results Max-Pooling

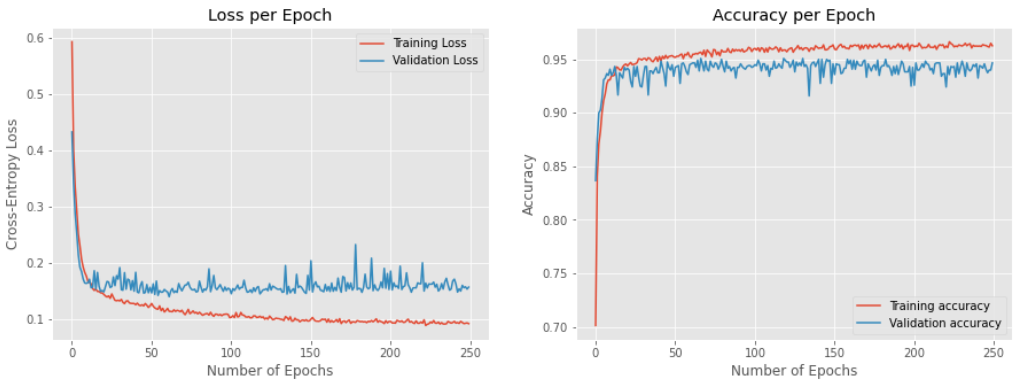

Figure 4.8: ANN Model Fit

Chapter 5

Discussion

This is the discussion section.

Chapter 6

Conclusion

This is the conclusion section.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M. et al. (2016), 'Tensorflow: Large-scale machine learning on heterogeneous distributed systems', *arXiv preprint arXiv:1603.04467* .
URL: *https://arxiv.org/abs/1603.04467*

Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V. & Smola, A. J. (2013), Distributed large-scale natural graph factorization, *in* 'Proceedings of the 22nd international conference on World Wide Web', pp. 37–48.
URL: *https://doi.org/10.1145/2488388.2488393*

Alphabet (2021), 'Alphabet investor relations'. (accessed: 04.05.2021).
URL: *https://abc.xyz/investor/*

AlphaFold (2020), 'Alphafold: a solution to a 50-year-old grand challenge in biology'. (accessed: 05.05.2021).
URL: *https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology*

Artisan's Asylum (2020), 'Intro to machine learning'. (accessed: 24.06.2021).
URL: *https://artisansasylum.com/classes/intro-to-machine-learning-2/*

Barabási, A.-L. & Albert, R. (1999), 'Emergence of scaling in random networks', *science* **286**(5439), 509–512.
URL: *https://doi.org/10.1126/science.286.5439.509*

Breiman, L. (2001), 'Random forests', *Machine learning* **45**(1), 5–32.
URL: *https://doi.org/10.1023/A:1010933404324*

Chang, C.-C. & Lin, C.-J. (2011), 'Libsvm: a library for support vector machines', *ACM transactions on intelligent systems and technology (TIST)* **2**(3), 1–27.
URL: *https://doi.org/10.1145/1961189.1961199*

Cohen, E. (2021), 'Node2vec', `https://github.com/eliorc/node2vec.git`.

Cramer, J. S. (2002), 'The origins of logistic regression'.
URL: *http://dx.doi.org/10.2139/ssrn.360300*

da Costa-Luis, C., Larroque, S., Altendorf, K., Mary, H., Korobov, M., Yorav-Raphael, N. et al. (2021), 'tqdm: A fast, extensible progress bar for python and cli', *Zenodo. Apr* .
URL: *https://doi.org/10.5281/zenodo.595120*

Dubois, Y. (2019), 'Interpretation of symmetric normalised graph adjacency matrix?', Mathematics Stack Exchange.
URL: *https://math.stackexchange.com/q/3284214*

Erdös, P. & Rényi, A. (1959), 'On random graphs 1', *Publicationes Mathematicae* **6**, 290–297.
URL: *https://snap.stanford.edu/class/cs224w-readings/erdos59random.pdf*

Euler, L. (1736), 'Solutio problematis ad geometriam situs pertinentis', *Commentarii academiae scientiarum Petropolitanae* **8**, 128–140.

Facebook (2021), 'Facebook investor relations'. (accessed: 04.05.2021).
URL: *https://investor.fb.com/investor-events/default.aspx*

Frauenhofer Institut (2021), 'Fame ai and machine learning'. (accessed: 24.06.2021).
URL: *https://www.fokus.fraunhofer.de/en/fame/workingareas/ai*

Freund, Y. & Schapire, R. E. (1997), 'A decision-theoretic generalization of on-line learning and an application to boosting', *Journal of computer and system sciences* **55**(1), 119–139.
URL: *https://doi.org/10.1006/jcss.1997.1504*

Grover, A. & Leskovec, J. (2016), node2vec: Scalable feature learning for networks, *in* 'Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining', pp. 855–864.
URL: *https://doi.org/10.1145/2939672.2939754*

Hagberg, A., Swart, P. & S Chult, D. (2008), Exploring network structure, dynamics, and function using networkx, Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).
URL: *https://www.osti.gov/biblio/960616*

Hamilton, W. L., Ying, R. & Leskovec, J. (2017), Inductive representation learning on large graphs, *in* 'Proceedings of the 31st International Conference on Neural Information Processing Systems', NIPS'17, Curran Associates Inc., Red Hook,

NY, USA, p. 1025–1035.
URL: *https://dl.acm.org/doi/abs/10.5555/3294771.3294869*

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T. E. (2020), 'Array programming with NumPy', *Nature* **585**(7825), 357–362.
URL: *https://doi.org/10.1038/s41586-020-2649-2*

Hastie, T., Rosset, S., Zhu, J. & Zou, H. (2009), 'Multi-class adaboost', *Statistics and its Interface* **2**(3), 349–360.
URL: *https://dx.doi.org/10.4310/SII.2009.v2.n3.a8*

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.
URL: *https://doi.org/10.1162/neco.1997.9.8.1735*

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

KAGGLE Inc., T. K. (2020), '2015 north america airport satisfaction survey'.
URL: *https://www.kaggle.com/teejmahal20/airline-passenger-satisfaction*

Katz, L. (1953), 'A new status index derived from sociometric analysis', *Psychometrika* **18**(1), 39–43.
URL: *https://doi.org/10.1007/BF02289026*

Kim, M. & Leskovec, J. (2012), 'Multiplicative attribute graph model of real-world networks', *Internet mathematics* **8**(1-2), 113–160.
URL: *https://doi.org/10.1080/15427951.2012.625257*

Kingma, D. P. & Ba, J. (2015), Adam: A method for stochastic optimization, *in* 'Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015'.
URL: *http://arxiv.org/abs/1412.6980*

Kipf, T. (2016), 'Graph convolutional networks'. (accessed: 15.05.2021).
URL: *https://tkipf.github.io/graph-convolutional-networks/*

Kipf, T. N. & Welling, M. (2016), 'Semi-supervised classification with graph convolutional networks', *arXiv preprint arXiv:1609.02907* .
URL: *https://arxiv.org/abs/1609.02907*

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), 'Imagenet classification with deep convolutional neural networks', *Advances in neural information processing systems* **25**, 1097–1105.
URL: *https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf*

Landau, E. (1895), 'Zur relativen wertbemessung der turnierresultate', *Deutsches Wochenschach* **11**, 366–369.

Leskovec, J. (2021), 'Cs224w lecture at stanford'. (accessed: 15.05.2021).
URL: *http://web.stanford.edu/class/cs224w/*

Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C. & Ghahramani, Z. (2010), 'Kronecker graphs: an approach to modeling networks.', *Journal of Machine Learning Research* **11**(2).
URL: *https://www.jmlr.org/papers/volume11/leskovec10a/leskovec10a.pdf*

Li, Y., Vinyals, O., Dyer, C., Pascanu, R. & Battaglia, P. (2018), 'Learning deep generative models of graphs', *arXiv preprint arXiv:1803.03324* .
URL: *https://arxiv.org/abs/1803.03324*

McCulloch, W. S. & Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133.
URL: *https://doi.org/10.1007/BF02478259*

McKinney, W. et al. (2010), Data structures for statistical computing in python, *in* 'Proceedings of the 9th Python in Science Conference', Vol. 445, Austin, TX, pp. 51–56.
URL: *http://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf*

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013a), 'Efficient estimation of word representations in vector space', *arXiv preprint arXiv:1301.3781* .
URL: *https://arxiv.org/abs/1301.3781*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. (2013b), 'Distributed representations of words and phrases and their compositionality', *arXiv preprint arXiv:1310.4546* .
URL: *https://arxiv.org/abs/1310.4546*

Moro, S., Cortez, P. & Rita, P. (2014), 'A data-driven approach to predict the success of bank telemarketing', *Decision Support Systems* **62**, 22–31.
URL: *https://doi.org/10.1016/j.dss.2014.03.001*

Moro, S., Laureano, R. & Cortez, P. (2011), 'Using data mining for bank direct marketing: An application of the crisp-dm methodology'.
URL: *http://hdl.handle.net/1822/14838*

Newman, M. (2010), *Networks: An Introduction*, Oxford University Press, Inc.

Newman, M. E., Barabási, A.-L. E. & Watts, D. J. (2006), *The structure and dynamics of networks.*, Princeton university press.

OECD (2017), 'G20/oecd infe report on adult financial literacy in g20 countries'. (accessed: 10.04.2021).
URL: *https://www.oecd.org/finance/g20-oecd-infe-report-adult-financial-literacy-in-g20-countries.htm*

Page, L., Brin, S., Motwani, R. & Winograd, T. (1999), The pagerank citation ranking: Bringing order to the web., Technical report, Stanford InfoLab.
URL: *http://ilpubs.stanford.edu:8090/422/*

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. et al. (2019), 'Pytorch: An imperative style, high-performance deep learning library', *arXiv preprint arXiv:1912.01703* .
URL: *https://arxiv.org/abs/1912.01703*

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011), 'Scikit-learn: Machine learning in python', *the Journal of machine Learning research* **12**, 2825–2830.
URL: *https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf*

Perozzi, B., Al-Rfou, R. & Skiena, S. (2014), Deepwalk: Online learning of social representations, *in* 'Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining', pp. 701–710.
URL: *https://doi.org/10.1145/2623330.2623732*

Platt, J. et al. (1999), 'Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods', *Advances in large margin classifiers* **10**(3), 61–74.
URL: *http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1639*

Power, J. (2015), '2015 north america airport satisfaction survey'.
URL: *https://www.jdpower.com/business/press-releases/2015-north-america-airport-satisfaction-study*

Schweitzer, F., Fagiolo, G., Sornette, D., Vega-Redondo, F., Vespignani, A. & White, D. R. (2009), 'Economic networks: The new challenges', *science* **325**(5939), 422–425.
URL: *https://doi.org/10.1126/science.1173644*

Seabold, S. & Perktold, J. (2010), Statsmodels: Econometric and statistical modeling with python, *in* 'Proceedings of the 9th Python in Science Conference', Vol. 57, Austin, TX, p. 61.
URL: *http://conference.scipy.org/proceedings/scipy2010/pdfs/seabold.pdf*

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W., Bridgland, A. et al. (2020), 'Improved protein structure prediction using potentials from deep learning', *Nature* **577**(7792), 706–710.
URL: *https://doi.org/10.1038/s41586-019-1923-7*

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. & Mei, Q. (2015), Line: Large-scale information network embedding, *in* 'Proceedings of the 24th international conference on world wide web', pp. 1067–1077.
URL: *https://doi.org/10.1145/2736277.2741093*

Tharwat, A. (2016), 'Linear vs. quadratic discriminant analysis classifier: a tutorial', *International Journal of Applied Pattern Recognition* **3**(2), 145–180.
URL: *https://doi.org/10.1504%2FIJAPR.2016.079050*

Van Rossum, G. & Drake, F. L. (2009), *Python 3 Reference Manual*, CreateSpace, Scotts Valley, CA.

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y. et al. (2019), 'Deep graph library: A graph-centric, highly-performant package for graph neural networks', *arXiv preprint arXiv:1909.01315* .
URL: *https://arxiv.org/abs/1909.01315*

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

Watts, D. J. & Strogatz, S. H. (1998), 'Collective dynamics of 'small-world'networks', *nature* **393**(6684), 440–442.
URL: *https://doi.org/10.1038/30918*

Xu, K., Hu, W., Leskovec, J. & Jegelka, S. (2018), How powerful are graph neural networks?, *in* 'Proceedings of the 7th International Conference on Learning Representation, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019'.
URL: *https://arxiv.org/abs/1810.00826*

You, J., Ying, R., Ren, X., Hamilton, W. & Leskovec, J. (2018), Graphrnn: Generating realistic graphs with deep auto-regressive models, *in* 'International Conference on Machine Learning', PMLR, pp. 5708–5717.
URL: *http://proceedings.mlr.press/v80/you18a.html*

You, J., Ying, Z. & Leskovec, J. (2020), 'Design space for graph neural networks', *Advances in Neural Information Processing Systems* **33**.
URL: *https://proceedings.neurips.cc/paper/2020/file/c5c3d4fe6b2cc463c7d7ecba17cc9de7-Paper.pdf*

Zhang, H. (2004), The optimality of naive bayes, *in* 'Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004, Miami Beach, FL, USA, May 12-14, 2004'.
URL: *https://www.aaai.org/Papers/FLAIRS/2004/Flairs04-097.pdf*

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C. & Sun, M. (2020), 'Graph neural networks: A review of methods and applications', *AI Open* **1**, 57–81.
URL: *https://doi.org/10.1016/j.aiopen.2021.01.001*