Master Thesis

Gaining Customer Insights using Machine Learning on Graphs

University of Basel

Author:
Michael von Siebenthal

Supervisor:
Prof. Dr. Dietmar Maringer

June 23, 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: June 23, 2021

Contents

1 Introduction **6**
- 1.1 Relevance to Economics . . . 6
- 1.2 Research Topic . . . 7
- 1.3 Research Question . . . 8
- 1.4 Literature Review . . . 8

2 Theory **9**
- 2.1 Graph Theory . . . 9
- 2.2 Machine Learning on Graphs . . . 14
  - 2.2.1 Graph Representation Learning . . . 14
  - 2.2.2 Graph Neural Networks . . . 18
    - 2.2.2.1 Graph Convolutional Networks . . . 21
    - 2.2.2.2 GraphSage . . . 23
- 2.3 Graph Generation . . . 25

3 Data **29**
- 3.1 Software . . . 29
- 3.2 Self Launched Survey . . . 30
- 3.3 Bank Telemarketing Dataset . . . 31
- 3.4 Airline Passenger Satisfaction Survey . . . 34
  - 3.4.1 Graph Generation . . . 36
- 3.5 Stochastic vs. Deterministic MAG . . . 41

4 Results **44**
- 4.1 Graph Representation Learning . . . 44
- 4.2 Graph Neural Networks . . . 46
  - 4.2.1 Graph Convolutional Network . . . 46
  - 4.2.2 GraphSage . . . 48
  - 4.2.3 GraphSage Robustness Simulation . . . 51
  - 4.2.4 Result Comparison . . . 52

5 Discussion **55**

6 Conclusion **56**

List of Figures

1.1 Bank Network 7

2.1 Example of a Graph 10
2.2 Network Embedding 15
2.3 Skip-Gram Architecture 17
2.4 GNN Structure 20
2.5 GraphSage Sampling 23
2.6 Node-attribute link-affinities 26
2.7 Schematic representation of the multiplicative attribute graphs (MAG) model 27

3.1 MAG graph of bank telemarketing dataset 31
3.2 Biased MAG graph of bank telemarketing dataset 33
3.3 Correlation Heatmap of US Airline Passenger Dataset 36
3.4 Graph of US Airline Passenger Dataset 38
3.5 Graph Nodes of US Airline Passenger Dataset 39
3.6 Graph Statistics 40
3.7 Deterministic MAG graph 42

4.1 Node2Vec embeddings 45
4.2 GCN Loss- and Accuracy Plots 47
4.3 Mean Aggregation Loss- and Accuracy Plots 49
4.4 LSTM Aggregation Loss- and Accuracy Plots 49
4.5 Sum-Pooling Aggregation Loss- and Accuracy Plots 50
4.6 Max-Pooling Aggregation Loss- and Accuracy Plots 51
4.7 Simulation Results Max-Pooling 54
4.8 ANN Model Fit 54

List of Tables

3.1 Confusion Matrix Validation Bank Telemarketing Data 32
3.2 Airline Dataset overview 35
3.3 Link Affinity Matrices 37

4.1 Node2Vec Classification Results 45
4.2 Confusion Matrix Training Data 47
4.3 Confusion Matrix Validation Data 47
4.4 Test Confusion Matrix Mean Aggregation 49
4.5 Test Confusion Matrix LSTM Aggregation 50
4.6 Test Confusion Matrix Sum-Pooling 50
4.7 Test Confusion Matrix Max-Pooling 51
4.8 Average Simulation Results 52
4.9 Result Comparison 53

Chapter 1

Introduction

The aim of this thesis is to explore the relatively new field of machine learning on graphs. In particular, this thesis will focus on graph machine learning applications for the purpose of gaining customer insights. Graph machine learning is the 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 model interactions or relationships.

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 of international banks as a network, see figure 1.1. This is a useful representation of interdependencies and is an important basis for making systems more robust.

Another interesting application of graphs for business & economics are social interactions. While there are many different types of social interactions of interest, social interactions for marketing purposes have been among the most popular. Indeed, this is one of the main areas where social networks such as Facebook or search providers such as Google make their revenue by selling advertising (Facebook 2021, Alphabet 2021). Both Facebook and Google have the advantage, that their busi-

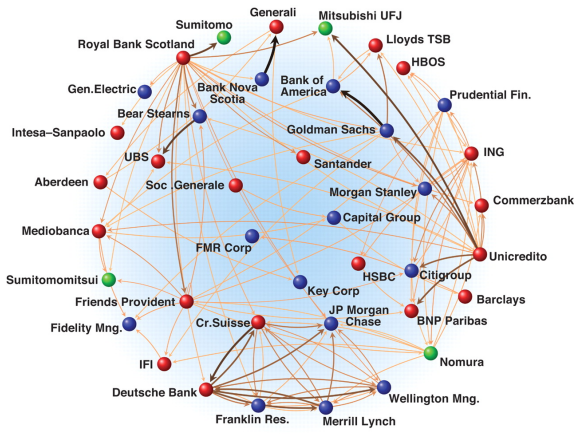

Figure 1.1: Bank Network
(Schweitzer et al. 2009, p. 424)

nesses 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. 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. The feature data of the people present in the network is however typically not provided. This is an issue in terms of data access.

1.2 Research Topic

Given the difficult access to graph data, this thesis will explore to what extent synthetic graphs can be generated using real-world cross-sectional data. An example for such data could be the client database of a company. The aim is then not only to generate graphs but to test whether the resulting graph can be used for meaningful machine learning tasks. In order to test this, appropriate datasets will be selected. It would of course be best, if one would have access to real graph data. If graph data had been available for an appropriate application, this would have been preferred for this thesis. The absence of such available graph data and the difficulty of generating / collecting real graph data sparked the interest and research topic for exploring synthetic graph generation for subsequent machine learning tasks. If this application proves to be successful, this could provide an alternative and hopefully successful approach for analyzing cross-sectional data. In order to assess, whether

synthetic graphs based on real cross-sectional data is a viable approach, the results of graph machine learning methods will be compared to "standard" machine learning methods. This leads to the research question formally presented in the following section.

1.3 Research Question

The research question for this thesis is defined as follows:

> **To what extent are synthetic graphs based on real cross-sectional data useful for machine learning tasks?**

In terms of machine learning task, this thesis will focus on a classification task. This ensures that the results of the graph machine learning methods versus the "standard" machine learning tasks can be compared.

1.4 Literature Review

An extensive literature review was conducted. To the best knowledge of the author, there are no other published studies which investigate the applicability of synthetic graphs based on real cross-sectional data for the purpose of machine learning. The literature review revealed the following related literature.

A large related research area is regarding synthetic graph generation. Classical models include the famous Erdös-Rényi graph generation model (Erdös & Rényi 2011), the small-world model by Watts and Strogatz (1998) or the model by Barabási & Albert (1999). More recent approaches include Kronecker Graphs (Leskovec et al. 2010) and its genralization Multiplicative Attribute Graphs (MAG) (Kim & Leskovec 2012). MAG work well in that it follows commonly observed network properties observed in real networks as outlined in the paper by Kim & Leskovec (2012). For the purpose of this thesis the MAG model will be used for transforming cross-sectional data into a graph. A more detailed introduction of the MAG model will be given in the theory section. Recently, synthetic graph generation models have shifted to deep generative models. Prominent models include GraphRNN by You et al. (2018) or the deep generative model presented by Li et al. (2018). These models differ to the traditional graph generation models in that they learn to create synthetic graphs based on real graph samples. These newer approaches could for instance be used for drug molecule discovery among many other approaches. These newer models are however not purposeful for this thesis.

Chapter 2

Theory

This chapter will cover the necessary theoretical background for this master thesis and consists of following 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 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. Vertexes are often referred to as nodes and the terms will be used interchangeably.

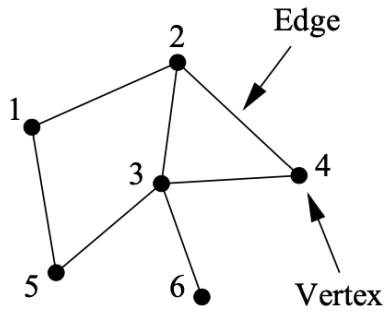

Figure 2.1: Example of a Graph
(Newman 2010, p. 111)

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 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 correspond 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 significantly different compared to traditional types of data usually used for machine learning tasks or regression analyses. 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 and 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

In the following subsections, methods for both graph machine learning categories will be introduced.

2.2.1 Graph Representation Learning

The aim of graph representation learning is to embed nodes into a d-dimensional vector representation. The resulting vector embeddings can then be used for "standard"

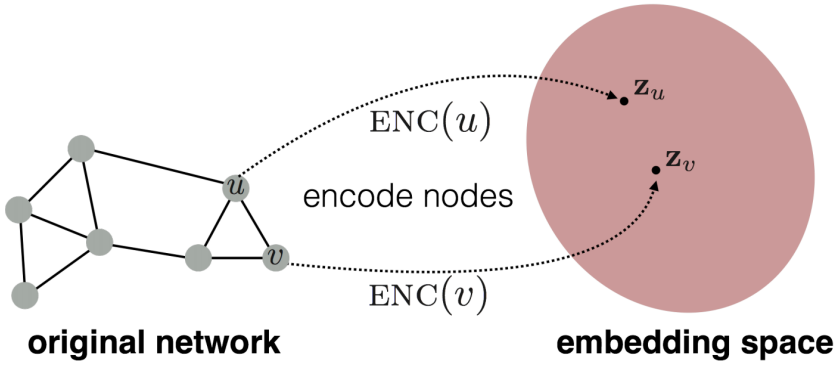

Figure 2.2: Network Embedding
Leskovec (2021)

machine learning applications. A graphical representation of this task is shown in figure 2.2.
Figure 2.2 outlines the task of finding an encoder such that nodes which are similar are also embedded close in the embedding space for which distance measures such as Euclidean distances can be calculated. A common measure for similarity is to find vector embeddings z of nodes u and node v such that (Leskovec 2021):

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

The dot product of two node vector embedding should thus approximately equal the similarity of the corresponding nodes in the network. There are different approaches for defining similarity of two nodes which can be applied. An early solution for instance was a graph factorization approach introduced by Ahmed et al. (2013). More recent and successful approaches include methods which make use of random walks. In this context, similarity is defined as follows (Leskovec 2021):

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

This thesis will focus on the methods Deepwalk by Perozzi et al. (2014) and its generalization Node2Vec by Grover & Leskovec (2016). There are other noteworthy methods such as LINE by Tang et al. (2015) which could be considered. In order to remain focused these additional methods are not considered. The selected methods are however well suited for the tasks given in this thesis.

DeepWalk and Node2Vec make use of methods which originated in natural language processing (NLP). Specifically they makes use of the Skip-Gram model introduced by Mikolov et al. (2013a,b). As the Skip-Gram model is a core component of Deep-

Walk and Node2Vec, the Skip-Gram framework is explained in detail in the following paragraph as it relates to graph representation learning.

In NLP words are one-hot encoded for the Skip-Gram model in which vector representations for the words are learned. Similarly, nodes in a network can be thought of as words in a NLP context. A basic overview of the Skip-Gram model is provided in figure 2.3. The input defined as X corresponds to a matrix with dimension $N \times N$ of one-hot encoded words or rather nodes. N corresponds to the number of nodes. The projection H is calculated using a simple forward propagation of the input times the weight matrix Φ with dimension $N \times D$. Here D defines the number of dimensions for the vector embedding of the words or nodes. The projection H is thereafter forward propagated again by multiplying it with a second weight matrix Ψ with dimensions $D \times V$. V refers to the context size a node is being compared to. For graph representation learning context size is defined by the number of neighboring nodes a node is being compared to. The output is thereafter activated using the Softmax activation function and the loss is calculated by comparing the activated output with the actual context node. The aim of the Skip-Gram model is to accurately predict the context of an input node, the neighbors of a given input node. The weights Ψ and Φ are thereafter updated using stochastic gradient descent. This procedure is repeated for several iterations and the Skip-Gram model is therefore a special form of a artificial neural network. The weight matrix Φ lastly corresponds to the node vector embeddings.

Based on this logic, the DeepWalk algorithm was developed which was a big breakthrough. The DeepWalk algorithm builds on top of the Skip-Gram model and uses 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 as defined in its original paper.

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 U^{|V|×d}
Build a binary Tree T from V
for i = 0 to γ do
    O = Shuffle(V)
    foreach v_i ∈ O do
        W_{vi} = RandomWalk(G, v_i, t)
        SkipGram(Φ, W_{vi}, w)
    end
end
```

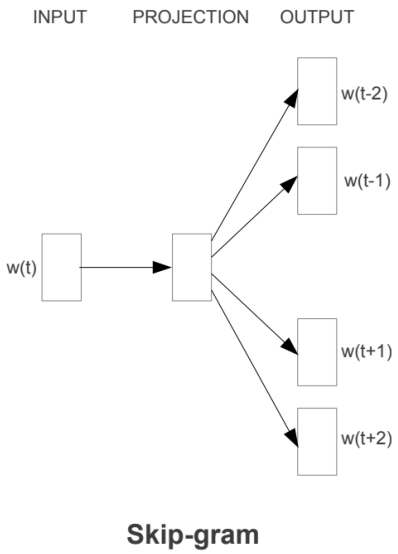

Figure 2.3: Skip-Gram Architecture
(Mikolov et al. 2013a, p. 5)

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 generates fixed length random walks for every node. The Skip Gram model is then run for every node on the random walk. In terms of context, the window size determines the neighboring nodes on the random walk the currently selected node is trying to predict in terms of context. The DeepWalk algorithm is then repeated for a defined number of random walks. In terms of training, the DeepWalk algorithm can lastly be run for a defined number of epochs. Lastly, the DeepWalk algorithm often uses more efficient approximation methods such as 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. In the application, the Node2Vec algorithm by will be employed. Node2Vec is a generalization of the DeepWalk algorithm and 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 (BFS) search 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 Node2Vec algorithm corresponds to the DeepWalk algorithm. More precisely, this occurs when the search bias $\alpha = 1$ 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.

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 will provide an overview regarding the theory about 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 graph G
- $X \in \mathbb{R}^{|V| \times F}$ is the matrix of 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 for instance be defined as a one-hot encoded vector. A naïve approach would be to join the adjacency matrix with the feature matrix as the input for a artificial neural network. The problem with this approach is, that the input is not order invariant and the 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 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 it neighbors. Further, node A is also a determining

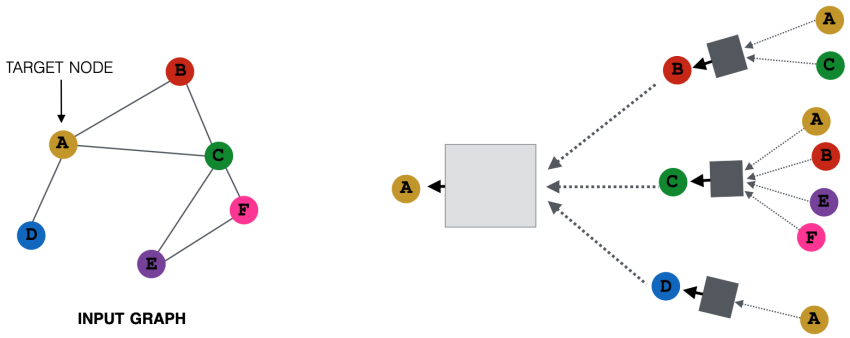

Figure 2.4: GNN Structure
Leskovec (2021)

factor for 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 training data

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

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

Initialize layer specific W^k from $\mathcal{U}, \forall k \in \{1, \dots, K\}$;
Initialize layer specific B^k from $\mathcal{U}, \forall k \in \{1, \dots, K\}$;
$h_v^0 = x_v, \forall v \in \mathcal{T}$
for *Number of epochs* **do**
  for $k = 1 \dots K$ **do**
    for $v \in \mathcal{T}$ **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
    $h_v^k \leftarrow h_v^k / \|h_v^k\|_2, \forall v \in \mathcal{T}$
  end
  $z_v = h_v^K, \forall v \in \mathcal{T}$;
  $\mathcal{L}(\Theta) = \sum_{(z_v, y_v) \in \mathcal{T}} CE(y_v, z_v)$;
  $\Theta = \Theta - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \Theta}$
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. 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 and the output of the final GNN layer corresponds to the node embedding of the target node A. This should sound familiar when comparing this approach to the graph representation learning approach 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 is 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. In order to avoid redundancy with the graph representation learning strategies introduced previously, the GNNs will be applied in a supervised learning context. GNNs are powerful tools for unsupervised learning for tasks such as clustering or graph representation learning (Zhou et al. 2020). While GNNs are possibly more efficient for graph representation learning (Kipf & Welling 2016, p. 12-13), the results are unlikely to differ significantly and is not the focus of this thesis. The aim is to use GNNs in a supervised learning setting as it hopefully provides superior results for a given classification task. In the following paragraph the first GNN method of interest is presented.

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. As outlined, GNNs typically differ with regards to the type of message passing and aggregation strategy. GCN make us of the following forward propagation method (Kipf & Welling 2016, p. 2):

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2.20}$$

The variables in equation 2.20 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.

To provide a better overview, the compact notation shown in equation 2.20 is expanded in the following equation for one GCN layer (Dubois 2019):

$$h_{ij}^{(l)} = \sigma\left(\sum_{(i,j)\in\mathcal{N}(v)} \frac{\tilde{a}_{ik} h_{kj}^{(l-1)}}{\sqrt{\tilde{d}_{k,k}\tilde{d}_{i,i}}} W^{(l)}\right) \tag{2.21}$$

In Equation 2.21 $h_{ij}^{(l)}$ 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):

$$Z = f(X, A) = \text{softmax}\left(\hat{A}\ \text{ReLu}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) \tag{2.22}$$

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 paragraph introduces GraphSage by Hamilton et al. (2017) which can be thought of as the inductive counterpart and generalization of the GCN presented in the previous paragraph. 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 unseen 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. In addition, GraphSage allows for efficiency increases when learning on large graphs by only considering a sample of the neighborhood $\mathcal{N}(v)$. In the GraphSage setting only a fixed number of uniformly sampled neighbors $\in \mathcal{N}(v)$ with depth K are considered. A graphical example of this procedure is shown in figure 2.5:

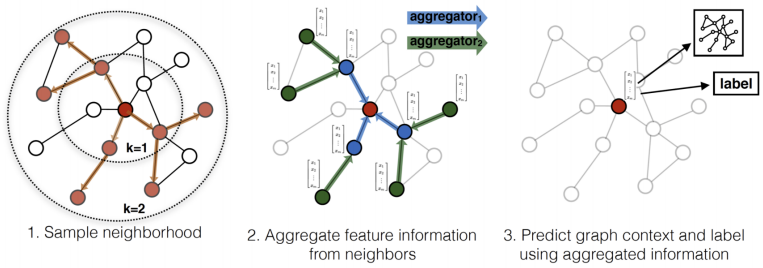

Figure 2.5: GraphSage Sampling
(Hamilton et al. 2017, p. 2)

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

Note that algorithm 4 assumes that the parameters of the K aggregator functions and the weight matrices W^k are known. Algorithm 4 thus shows the forward propa-

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, $\mathcal{N}_k : v \rightarrow 2^v, \forall k \in \{1, \dots, K\}$
Output: Vector representations z_v for all $v \in \mathcal{B}$

1 $\mathcal{B}^K \in \mathcal{B}$
2 **for** $k = K \dots 1$ **do**
3 $\quad \mathcal{B}^{k-1} \leftarrow \mathcal{B}^k$
4 $\quad$ **for** $u \in \mathcal{B}^k$ **do**
5 $\quad\quad \mathcal{B}^{k-1} \leftarrow \mathcal{B}^{k-1} \cup \mathcal{N}_k(u)$
$\quad$ **end**
end
6 $h_u^0 \leftarrow x_v, \forall v \in \mathcal{B}^0$
7 **for** $k = 1 \dots K$ **do**
8 $\quad$ **for** $u \in \mathcal{B}^k$ **do**
9 $\quad\quad h^k_{\mathcal{N}(u)} \leftarrow AGGREGATE_k(\{h^{k-1}_{u'}, \forall u' \in \mathcal{N}_k(u)\})$;
10 $\quad\quad h^k_u \leftarrow \sigma\left(W^k \cdot CONCAT(h^{k-1}_u, h^k_{\mathcal{N}(u)})\right)$;
11 $\quad\quad h^k_u \leftarrow h^k_u / \|h^k_u\|_2$
$\quad$ **end**
end
12 $z_v \leftarrow h^K_u, \forall u \in \mathcal{B}$

gation 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. In algorithm 4 $\mathcal{B}$ refers to the mini-batch training for 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):

$$h_v^k \leftarrow \sigma\left(W \cdot \text{MEAN}(\{h_v^{k-1}\} \cup \{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})\right) \tag{2.23}$$

Max Pooling:
Max Pooling aggregation refers to the application of an element-wise max operator. This means that of the neighbors, only the largest 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):

$$AGGREGATE_k^{pool} = \max\left(\sigma(\{W_{pool}h_{u_i}^k + b), \forall u_i \in \mathcal{N}(v)\}\right) \tag{2.24}$$

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:
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

As mentioned in the introduction, gathering network data including features is rather difficult task. This is especially true, as most databases do not capture relationships. Further, it is very difficult if not impossible to collect network data using standard surveys. This is a problem for this master thesis for which no perfect solution exists. As outlined, this problem sparked the research question as to whether network data can be generated from cross-sectional data for subsequent and meaningful machine learning tasks. Among the many graph generation algorithms researched, the Multiplicative Attribute Graph (MAG) model by Kim & Leskovec (2012) appears to provide a feasible solution and will be used for this thesis.

The MAG model makes use of two main inputs which are:

1. Node attribute vectors a_i
2. Link-affinity matrices Θ_i

The node attribute vector a_i is part of the graph node-attribute generation matrix $B^{N\times K}$. For the purpose of graph generation, only attributes can be considered for which reasonable link-affinity probabilities can be defined. Therefore $B \subseteq X$ where $X^{N\times F}$ is the full feature matrix with N observations and F features. The feature matrix X could for instance be data collected in a survey or a client database with absent relational information. F could for instance refer to the number of questions in a survey for which responses were collected. Every respondent in the survey subsequently is considered as a node in graph setting and the responses are the corresponding node features. Note, that the authors refer to node features as attributes. The terms are used interchangeably for the purpose of this thesis. The K appropriate features which can be included in B are variables for which reasonable link-affinity probabilities can be defined. A typical example for this could be age. One could imagine an appropriate setting in which people of the same age group are more likely to form a connection / become friends in a social network setting versus people which are in different age groups. Another example for this could be gender in terms of biological sex which is a classical binary setting for a link-affinity matrix. An example for binary attribute link-affinity matrices Θ_i is given in figure 2.6.

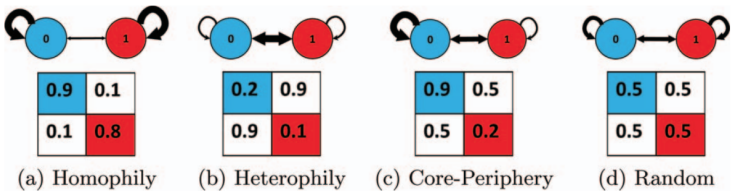

Figure 2.6: Node-attribute link-affinities
(Kim & Leskovec 2012, p. 118)

Figure 2.6 shows 4 types of link-affinity matrices depending on the relationship between nodes one wants to model. Homophily refers to love of the same which would make a connection between two nodes more likely if they have the same features. Similarly, Heterophily refers to the love of the different where nodes which do not have the same features 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). For instance one could model a group of people in terms of whether they are members in a football club or not. It would then be reasonable to assume, that members of a football club are very likely to be connected while non-members have a significantly lower probability of forming a connections. Lastly, random graphs can be generated by setting the link-affinity probabilities to 0.5. Given the available data sets, graphs will be generated using

homophily structures. The node-attribute link-affinity matrices Θ_i are defined for every attribute and can be set for an arbitrary size of categories within an attribute (extends beyond the binary setting). More formally for each node $u \in V$ with K categorical attributes of cardinality d_i for $i = 1, 2, \ldots, K$ and corresponding link-affinity matrices $\Theta_i \in d_i \times d_i$ for $i = 1, 2, \ldots, K$, the probability $P[u, v]$ of an edge between nodes (u, v) is defined as (Kim & Leskovec 2012, p. 119):

$$P[u, v] = \prod_{i=1}^{K} \Theta_i[a_i(u), a_i(v)] \tag{2.25}$$

In equation 2.25 $a_i(u)$ refers to the value of the ith attribute of node u. A schematic representation of the procedure for a binary link-affinity matrix is shown in figure 2.7.

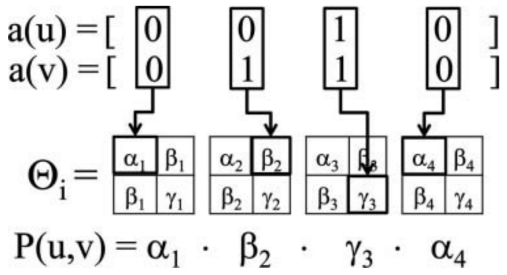

Figure 2.7: Schematic representation of the multiplicative attribute graphs (MAG) model

(Kim & Leskovec 2012, p. 120)

Algorithm 5 generates the adjacency matrix A for constructing the resulting graph $G(V, E)$. The node features X can thereafter be assigned to the corresponding nodes as the node ordering of the adjacency matrix corresponds to the ordering of the node feature matrix X. The resulting graph $G(V, E)$ is therefore constructed by first assigning the probabilities for two node (u, v) to form a connection based on the link-affinity matrices Θ_i. Once the probabilities are calculated, a connection is formed if the probability for a connection is larger than a randomly drawn value from a standard uniform distribution $\mathcal{U}(0, 1)$. If the probability is lower, no connection is formed. As this thesis focuses on undirected graphs with no self-edges, the resulting adjacency matrix A is symmetric with $diag(A) = 0$. In order to ensure this, the upper triangular matrix U of the probability matrix P is taken and setting the $diag(U) = 0$. Tis ensures, that the resulting adjacency matrix is symmetric and contains no self-edges.

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 $\Theta_i \in d_i \times d_i$ 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
$A = \hat{A} + \hat{A}^T$

Chapter 3

Data

This section introduces the datasets used for this thesis. Several approaches and datasets were considered for the subsequent analysis. In particular, three datasets were considered with varying degrees of success which are:

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

The datasets are introduced in the following subsections to the extent that they were of use for the purpose of this thesis. In particular, the self launched survey and the bank telemarketing dataset showed to be problematic for different reasons. These two datasets are therefore briefly introduced and the results which highlight the difficulties of the datasets are shown directly. The detailed analyses of these two datasets are deferred to the appendix as the focus will be placed on the airline passenger satisfaction survey for which more promising results were achieved.

In the upcoming subsections the different datasets are briefly introduced and reasons for its use or non-use will be given. First, the used programming language and packages are thankfully referenced in the following subsection.

3.1 Software

The entire thesis was evaluated using the Python 3 programming language (Van Rossum & Drake 2009). In addition, following open-source python packages were thankfully used which are Numpy (Harris et al. 2020), Matplotlib (Hunter 2007), NetworkX (Hagberg et al. 2008), Seaborn (Waskom 2021), Pandas (McKinney et al. 2010),

Statsmodels (Seabold & Perktold 2010), Scikit-Learn (Pedregosa et al. 2011), Tensorflow (Abadi et al. 2016), Pytorch (Paszke et al. 2019), deep graph library (dgl) (Wang et al. 2019), tqdm (da Costa-Luis et al. 2021) and Node2Vec (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). It is suspected, that demographic data coupled with the financial literacy questions should be provide a suitable database for the bank client classification task given. This is 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 graph. Classification accuracies ranged between accuracies of 40 - 95 %. A remedy for this problem would be to assign a fixed probability threshold such as 0.5 in the MAG model. In this setting, the graph generation would be deterministic, however it would ensure identical graph generation. The downside however is that the graph generation process is less realistic. In the homophily setting this would assume that a connection is formed with any node if the $P[u, v] > 0.5$. From our shared human experience we know, that people often form friendships or other connections with people who can differ significantly to themselves. This raises the question as to whether we should care about the stochastic element involved in forming connections for a synthetically created graph for the purpose of a classification task? This is an important question which was discovered due to the small sample size of the self created survey and will be discussed in a subsequent subsection. Due to the small sample size which makes the survey data inadequate for any meaningful machine learning task, the survey was finally discarded for further analysis for this thesis. In

Appendix X an overview of the survey data is given. Further, the performed analyses for this dataset are provided in the accompanying git repository. Nevertheless, the survey can be taken as a model of how one could generate a dataset for the MAG model and subsequent graph machine learning.

3.3 Bank Telemarketing Dataset

The bank telemarketing dataset first introduced in the article by Moro et al. (2011, 2014) was considered as an alternative back-up dataset in case 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, information regarding the success of contacting the client in previous campaigns and information such as the length of the telephone conversation. The dataset further provides information as to whether the campaign was successful in that the contacted client invested in a short-term deposit which was being advertised in the telemarketing campaign. This is the label data and the dataset is thus staged as a binary classification task. The resulting MAG graphs is shown in figure 3.1.

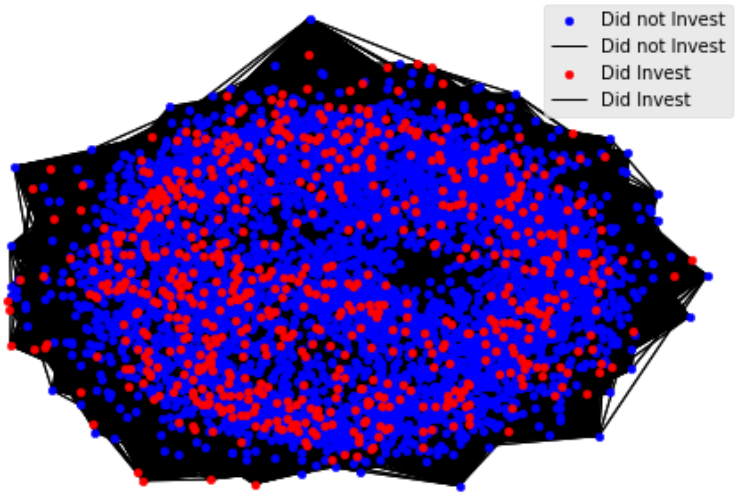

Figure 3.1: MAG graph of bank telemarketing dataset

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. While this figure masks some of the nodes due to the figure generation process, the general pattern becomes apparent. The red nodes appear to be randomly placed in the network which suggests, that the demographic variables which are suitable and were used for the MAG generation process do not capture the structure of the label well. 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, in the dataset only consists of approximately 12% of bank clients which invested in the short-term deposit. The dataset is therefore relatively unbalanced which makes the classification task rather difficult. Graph representation learning using Node2Vec did not provide any useful results and the GNNs also performed rather poorly. In particular, GNNs tendedto classify most clients as non-investors and struggled to accurately classify clients which did invest. Due to the unbalanced label data, it was loss optimizing to predict most nodes as non-investors rather than learning the true observations. Table 3.1 shows the confusion matrix of the model classifications for the validation data (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 graph generation and subsequent model fitting was repeated several times and the accuracies all ranged between 87-90%. This behavior was observed for both graph based methods and standard machine learning approaches such as ANN or support vector machines (SVM).

This is 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 problem as dropping a large number of observation 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. This thesis will not focus on this problem which is why this issue is not further investigated.

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 was used for the

MAG network generation process. Indeed by setting the link-affinity probabilities as follows for the label, superior results were achieved:

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

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

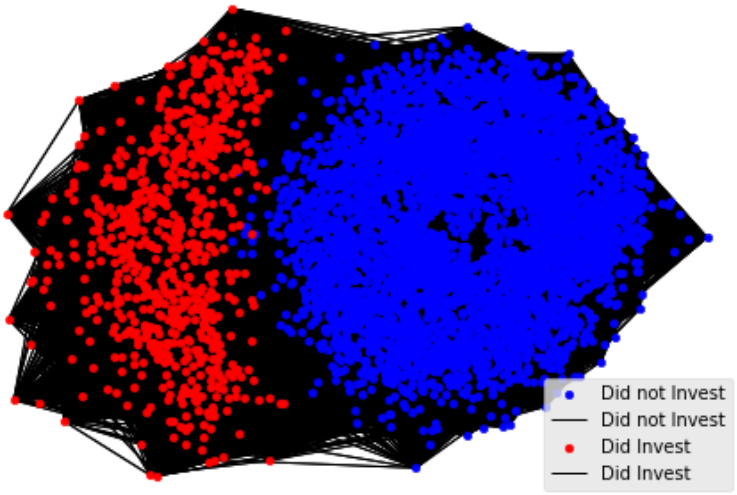

Figure 3.2: Biased MAG graph of bank telemarketing dataset

It now becomes apparent in figure 3.2 that the nodes are almost perfectly separated from each other and grouped according to their label. The GNN method GraphSage achieved an accuracy of over 95% for the graph shown in figure 3.2. This is of course a form of cheating, as we cannot assume to know the labels of the entire graph and especially when applying the model to new and unseen data which might not contain any label information. It however shows, that if the graph provides node clusters which corresponds to the node labels, that graph machine learning can overcome the problem of unbalanced data to a cerain extent. The difficulty for graph machine learning here is to find graph data including features which naturally captures node clusters which correspond to the node labels. For the synthetic graph generation setting using the MAG model, one would have to collect data which generates clusters via the MAG model that highly correlate with the label. An example for such a dataset was found and while not perfect, it yielded good results and is presented in the following section.

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. Therefore, this dataset is suitable for a fair comparison of graph machine learning methods vs standard machine learning strategies. This dataset will be presented in more detail as it will be used for the detailed analyses which are to follow.

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 model performed better if these variables were included in the model. The table with a brief description of the variables is provided on the following page. 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.

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
Class	Travel class in the plane of the passengers (Eco:1, Eco Plus:2, Business:3)	-	0 - 2
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
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

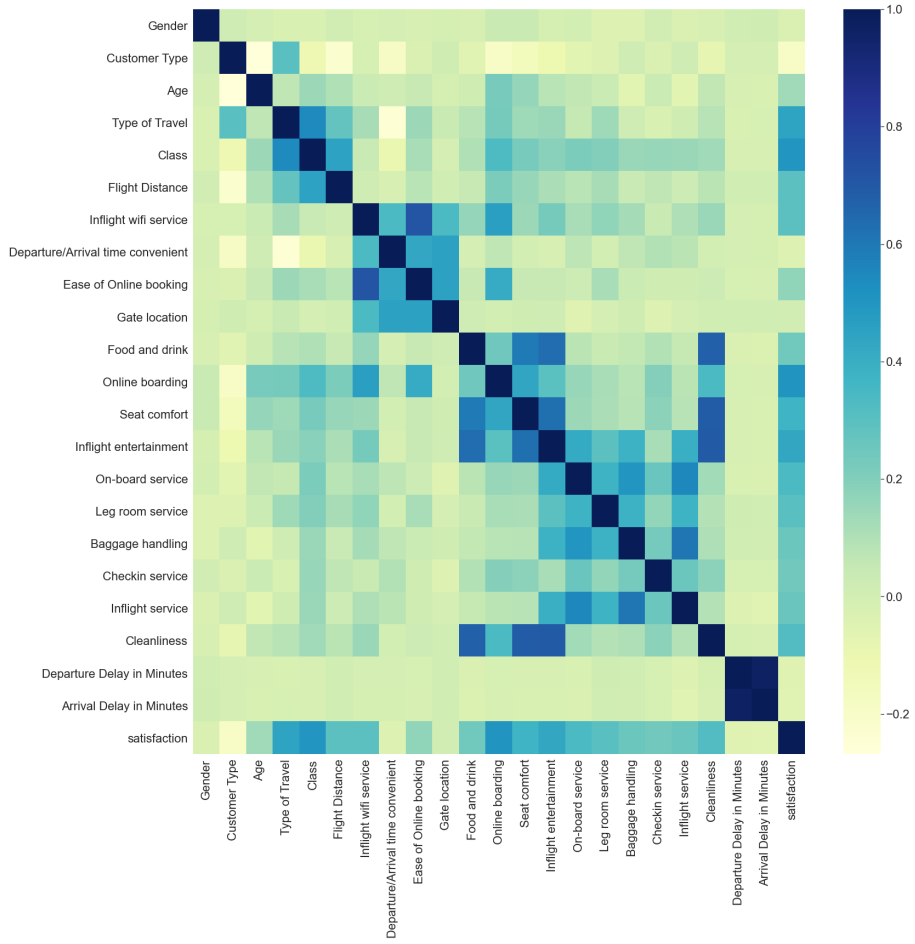

Figure 3.3: Correlation Heatmap of US Airline Passenger Dataset

The variables of the dataset can further be classified as follows:

- **Categorical Variables:** Gender, Class, Customer Type, Type of Travel, Satisfaction
- **Ordinal Variables:** 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 as shown in table 3.2. The remaining variables are already correctly in the dataset as is. In the following subsection the graph generation procedure for the US Airline Passenger dataset is briefly described.

3.4.1 Graph Generation

To create a graph from the US Airline Passenger Dataset, appropriate variables must be selected for the MAG model. The selected variables must be of the type

such that realistic probabilities can be applied in terms of homophily or heterophily. As an example it is for instance 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 than 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 variables for the graph generation more difficult. It is therefore important to select variables for which realistic variables for all 3 settings which are:

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

The variables were selected using the above mentioned considerations. The variables with the corresponding link-affinity probabilities are shown in table 3.3.

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

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:

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

Age was not only selected as an example as its 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 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 $\geqslant 50$. The value of the bin corresponds to the position of two observations (u, v) in the link-affinity matrix row/column. These bins were chosen according to the interquartile lengths present in the variable age.

For assigning probabilities there exist no rules as to how these must be applied. The paper by Kim & Leskovec (2012) shows the 4 common structures of homophily, heterophily, core-periphery and random for creating graphs. For this thesis, the homophily approach was selected where the more similar two observations are, the more likely they are to form a connection. For the selected variables reasonable similarity and dissimilarity probabilities could be assigned in a homophily setting. The probabilities were assigned based on personal intuition and experimentation. Several graphs were created with different probabilities and the probabilities shown in table 3.3 appear to generate reasonable graphs. There is however no exact science or selection criteria applied when choosing the probabilities.

A random sub-sample of 6'000 observations was retrieved from the training dataset consisting of 103'904 observations. With this random sub-sample the adjacency matrix for the resulting graph was generated using algorithm 5 with the link-affinity matrices shown in table 3.3. The random sub-sample was retrieved due to the computational cost involved applying algorithm 5. Simulations involving creating graphs with different random sub-samples suggest, that the random sub-samples are representative for the entire dataset. The resulting graph used for the subsequent analyses is shown in figure 3.4.

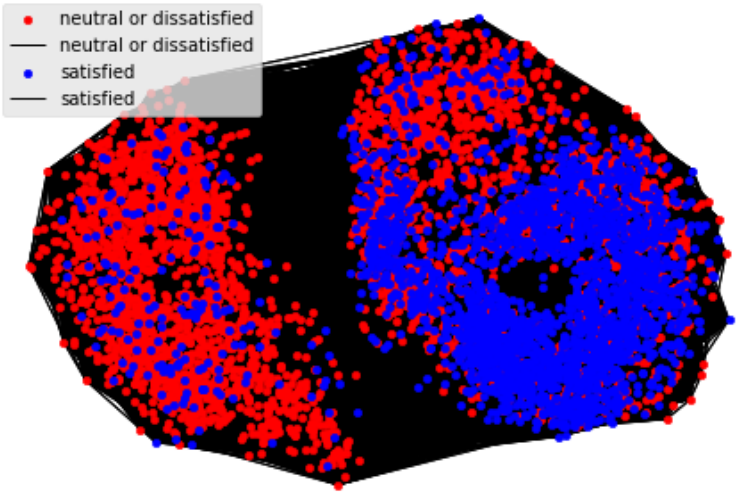

Figure 3.4: Graph of US Airline Passenger Dataset

The network shown in figure 3.4 show the emergence of two groups. In addition one can see, that most satisfied airline passengers appear to be somewhat clustered

together. To gain a deeper understanding of the dynamics involved in the network formation, the nodes of the network (excluding the edges) are plotted in figure 3.5.

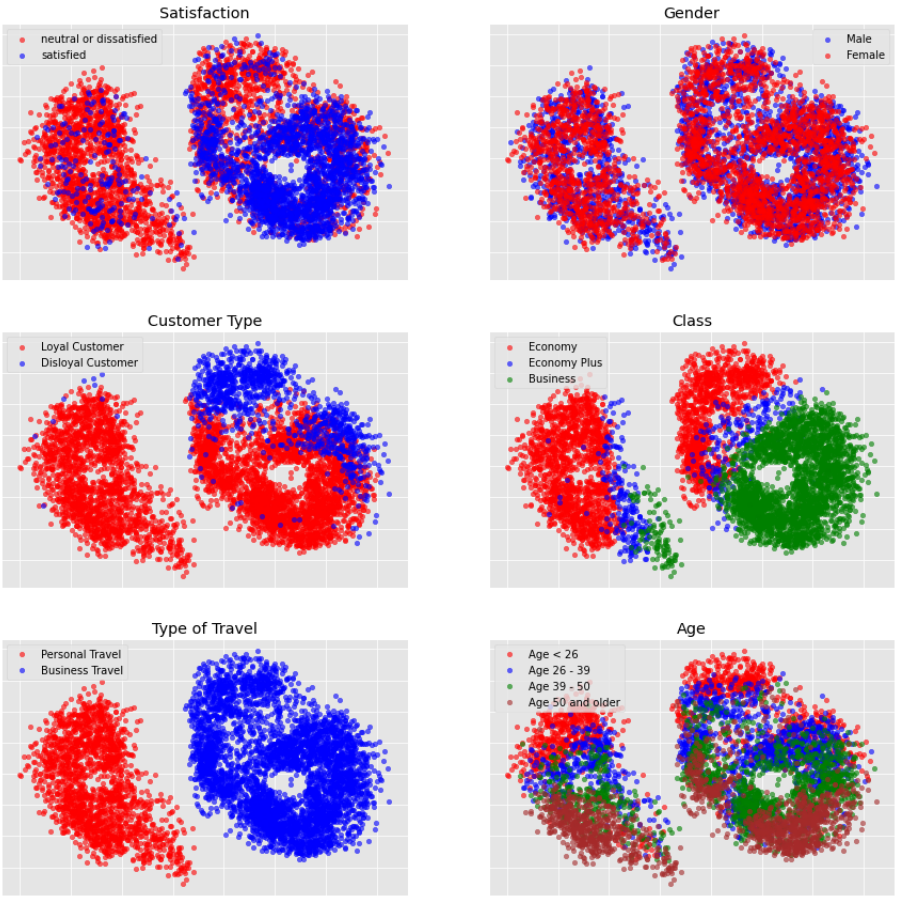

Figure 3.5: Graph Nodes of US Airline Passenger Dataset

Figure 3.5 plots the nodes of the network for the label "Satisfaction" and the 5 variables used for the graph generation. The nodes were plotted with $\alpha = 0.6$ to avoid covering nodes during the graph generation process. The node plots reveal interesting relationships. First, we can see that people traveling for business purposes tend to be more satisfied than people traveling for personal reasons. On a second level we can see, that people who flew business class appear to be mostly satisfied and constitute the majority of the cluster with satisfied passengers visible on figures 3.4 & 3.5. When comparing to people traveling for personal reasons, "Class" does not appear to be a determining factor for satisfaction. For the variable "Age" we can see that older people tend to fly more business class when on a business trip. In addition, older people traveling for business purposes tend to be loyal customers compared to younger customers. Interestingly, these relationships do appear to be present for passengers traveling for personal reasons. Interestingly, most passenger traveling for personal reasons appear to be loyal customers. This however does not appear to translate into positive satisfaction. Lastly, the variable Gender does not

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

Last but not least, the graph is described using metrics from graph theory. 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.

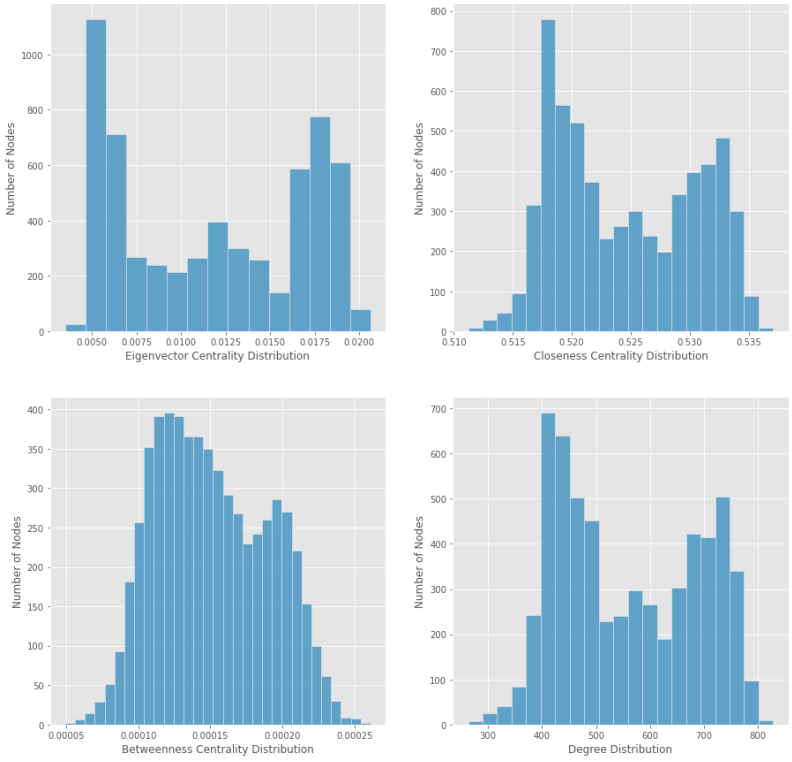

Figure 3.6: Graph Statistics

The network further has a density of approximately 0.0933. This means that 9% of the maximum 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. 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 typically 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 ovserved in real graphs. 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 revealed that the graph is indeed useful for graph machine learining. In addition, limiting factors of the graph were discovered. These aspects will be discussed further in the results and discussion section.

3.5 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 setting 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 to 0.5, not a single connection between observations was made. This makes sense as the probability of a connection being formed decreases by design as the number of generation variables increases. This is implicitly shown in equation 2.25 where the product of probabilities is bound to decrease. For this reason, it is suggested to set $K = \rho \log_2 N$ for some constant ρ (Kim & Leskovec 2012, p. 122). For the purpose of this thesis, the generation variables were selected such that $K \leqslant \log_2 N$. The airline passenger satisfaction survey was used for a deterministic graph generation. The threshold probability for a connection was set to 0.2 in the MAG model. The resulting graph consisted of several disconnected sub-graphs which were for the most part clustered according to their group membership. Figure 3.7 shows the respective plots of the graph generation variables where the edge connections were removed. The sub-graphs are unfortunately small, however all graphs combined include all 6'000 nodes. The plots are meant to provide a high-level overview of the clusters in the sub-plots.

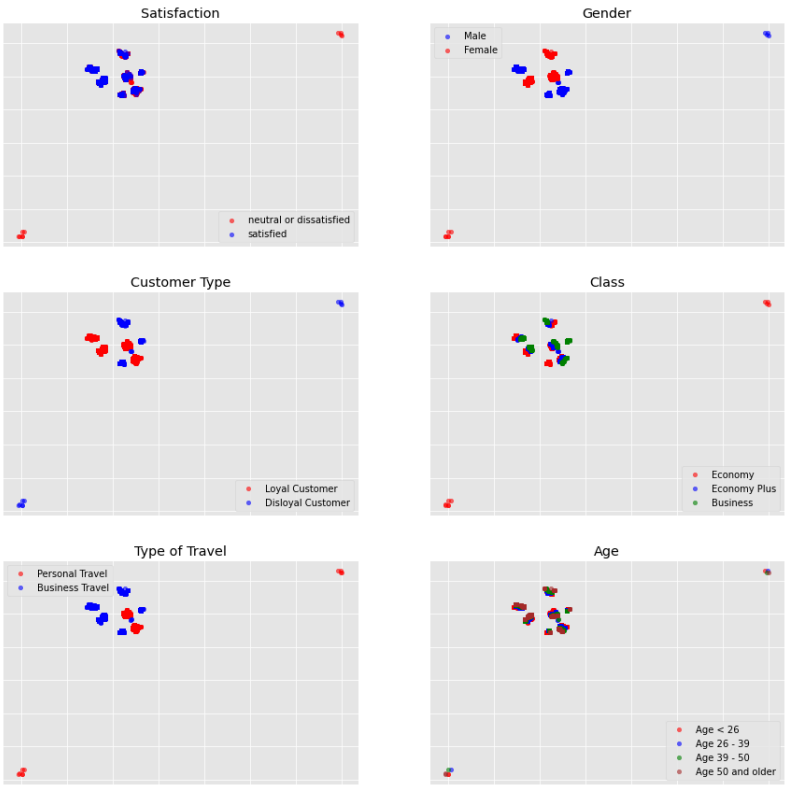

Figure 3.7: Deterministic MAG graph

In short, the deterministic graph clustering creates disconnected sub-graphs which forms clusters based on the nodes similarity to each other. In terms of performance, the performance and loss behavior of both the deterministically and stochastically generated graphs are virtually identical. This was true for the datasets considered for this thesis. For the purpose of visualization the stochastically generated graph

appears 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 sub-graphs. These sub-graphs or clusters could then be used for subsequent machine learning tasks with a cluster specific target. The clusters further could reveal information as to which clusters tend to be more or less satisfied. This is an area which could be interesting for future research. It is however not the focus of this thesis which is why it is not further investigated.

Chapter 4

Results

In this section 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 of "standard" machine learning methods.

4.1 Graph Representation Learning

The training graph consisting of 6'000 nodes which is a sub-sample of the US Airline Passenger training dataset provided on Kaggle (KAGGLE Inc. 2020). 2 Dimensional node representations were learned using the Node2Vec algorithm (Grover & Leskovec 2016). The model settings were defined as follows:

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

With the specified Return- and In-out parameter, the Node2Vec output corresponds to the DeepWalk output. The resulting node embeddings were then used as inputs for standard machine learning methods. An overview of the node embeddings is shown in figure 4.1.
The embeddings reveal very interesting insights in term of cluster analysis. First, the observation seen in figure 3.4 mainly correspond to the observations in figure

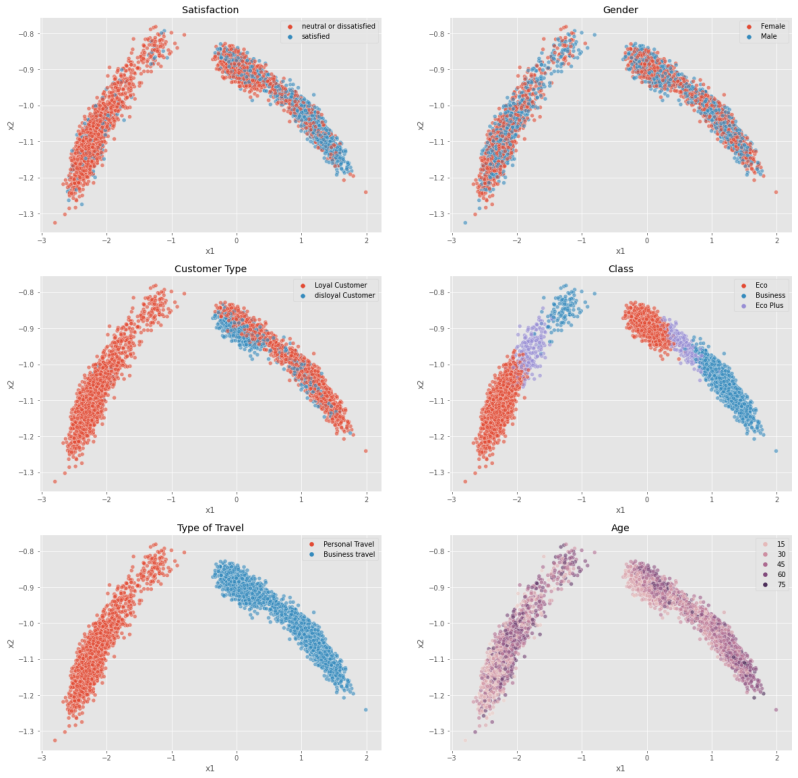

Figure 4.1: Node2Vec embeddings

4.1. The advantage of graph representation learning, is that the nodes plotted using Node2Vec corresponds to a proper scatter plot in Euclidean space. In addition, the nodes are group in a much nicer and more orderly fashion. This makes it possible to infer with a higher level of certainty that for instance business class passengers traveling for business purposes tend to be more satisfied than other passengers on average. The 2 dimensional node embeddings thus provide interesting insights for cluster analysis.

The node embeddings were used for three standard machine learning methods which are presented in table 4.1.

ML Method	**Training Accuracy**	**Validation Accuracy**
Logistic Regression	76.85%	75.83%
Support Vector Machine	76.95%	75.66%
ANN	77.61%	76.58%

Table 4.1: Node2Vec Classification Results

The results show, that Node2Vec only has very modest success in classifying passengers according to their satisfaction. When looking at the satisfaction scatter plot in figure 4.1 it becomes obvious why the downstream machine learning tasks only had limited success. For the variables which were used for generating the graph

and for which neighborhood clusters emerged, Node2Vec created very good node embeddings. The label variable satisfaction could not be used for the graph generation as this would be unrealistic. Since the label is not considered for the graph generation, Node2Vec does not create embeddings which consider the label. The label is only considered to the extend that the variables used for the graph generation process create structures which are related somehow to 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. The Python code for generating the node embeddings and the subsequent machine learning methods can be found in appendix X.

4.2 Graph Neural Networks

This subsection will present the results using graph neural networks. As mentioned in section 2, GCN by Kipf & Welling (2016) and GraphSage by Hamilton et al. (2017) will be used for classifying the satisfaction of the US airline passenger dataset. The model specifications and results are presented in the following subsections.

4.2.1 Graph Convolutional Network

The GCN was designed using the exact forward propagation function previously outlined in equation 2.22.

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

The node features X correspond to the 21 selected explanatory variables of the US airline passenger dataset as described in section 3. The node features were standardized due to large differences between the variables in terms of scale. In addition, standardizing yielded significant performance improvements. The hidden layer size for both convolutional layers was set to 21 and the output layer was set to size 2 for the binary classification. 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 the model parameters are updated with the Adam optimizer instead of standard gradient descent. Different model specification were tested, however the chosen specifications appeared to perform very well. For example, choosing a hidden layer size of 100 or 128 yielded virtually identical results. A stronger reduction in hidden layer size however reduces the performance of the GCN. Adding additional convolutional layers decreased the improvements and decreasing the learning rate decreased the performance. This

graph convolutional network requires approximately 1'200 epochs to finish training. The loss- and accuracy plots are shown in figure 4.2.

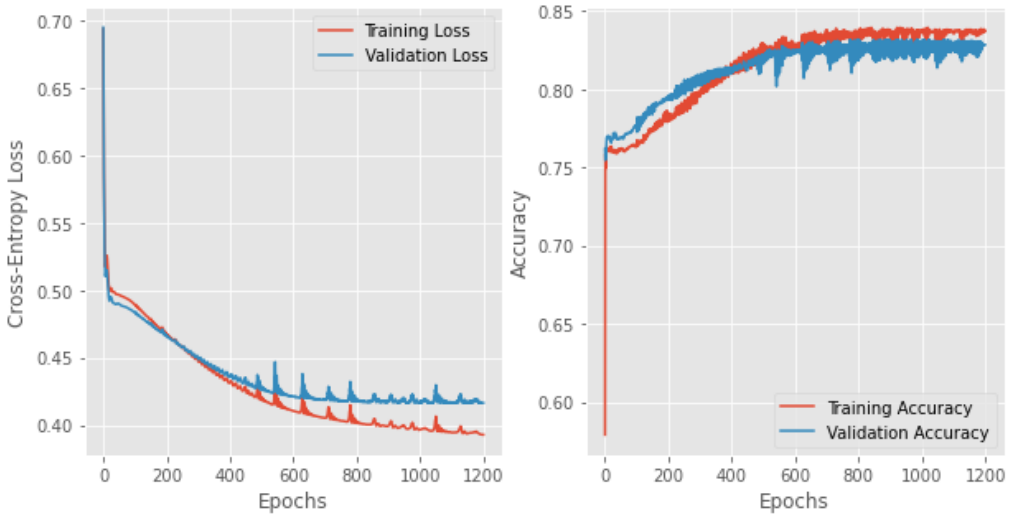

Figure 4.2: GCN Loss- and Accuracy Plots

The results are are further presented in table 4.2 & 4.3.

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	893	143
Satisfied	148	605
Accuracy	83.73%	

Table 4.2: Confusion Matrix Training Data

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	2'004	383
Satisfied	339	1'485
Accuracy	82.85%	

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.

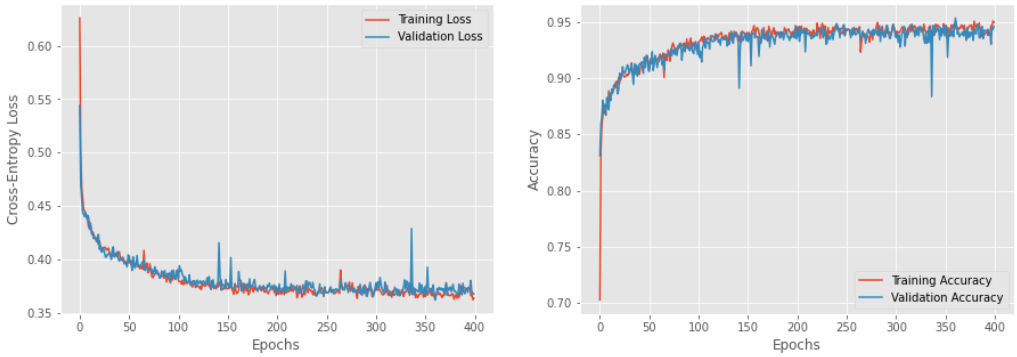

Figure 4.3: Mean Aggregation Loss- and Accuracy Plots

The training accuracy is 94.98% and the validation accuracy is 94.59% after 400 epochs. The model resulted in a test accuracy of 93.38% with the following confusion matrix shown in table 4.4:

Label \ Predicted	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	3'262	94
Satisfied	303	2'341
Accuracy	93.38%	

Table 4.4: Test Confusion Matrix Mean Aggregation

LSTM Aggregation

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

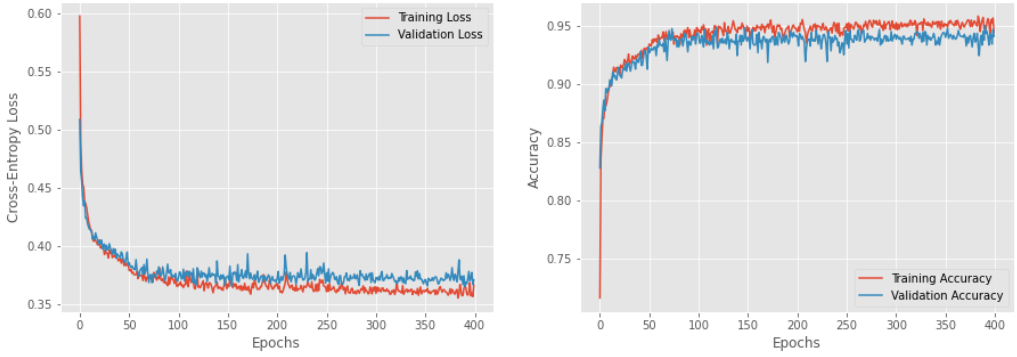

Figure 4.4: LSTM Aggregation Loss- and Accuracy Plots

The training- and validation accuracy after 400 epochs was 94.44% and 94.09% 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	3'259	97
Satisfied	295	2'349
Accuracy	93.47%	

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.

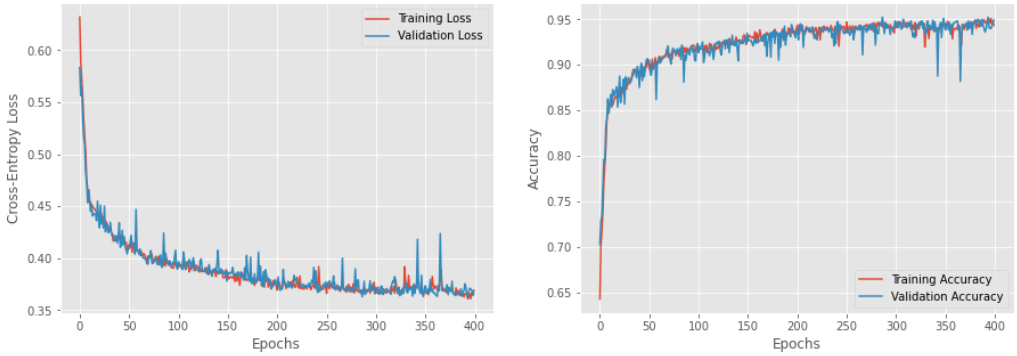

Figure 4.5: Sum-Pooling Aggregation Loss- and Accuracy Plots

The plots show that sum-pooling trains very well. The training accuracy is 94.33% and validation accuracy is 94.2% after the model finished training. The results for the test graph are shown in table 4.6.

Label \ Predicted	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	3'218	138
Satisfied	212	2'432
Accuracy	94.17%	

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.

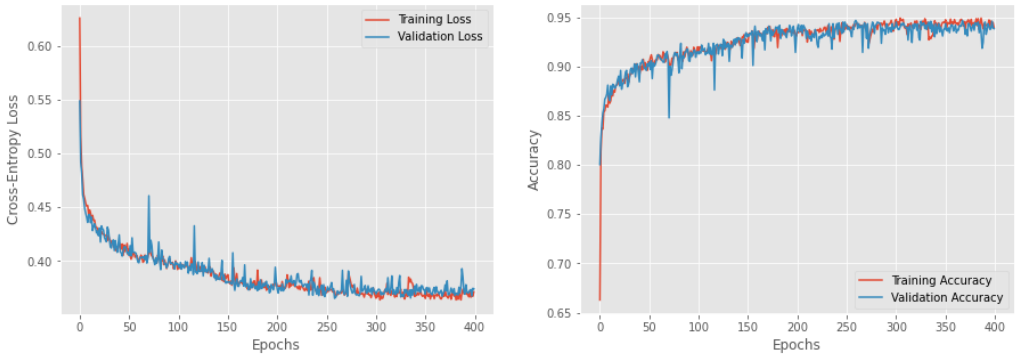

Figure 4.6: Max-Pooling Aggregation Loss- and Accuracy Plots

The training model resulted with a validation accuracy of 93.96% and training accuracy of 93.84%. The results for the test graph are shown in table 4.7.

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	3'180	176
Satisfied	196	2'448
Accuracy	93.8%	

Table 4.7: Test Confusion Matrix Max-Pooling

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 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.

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

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

4.2.4 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. 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) and an Artificial Neural Network (McCulloch & Pitts 1943) are used. 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.

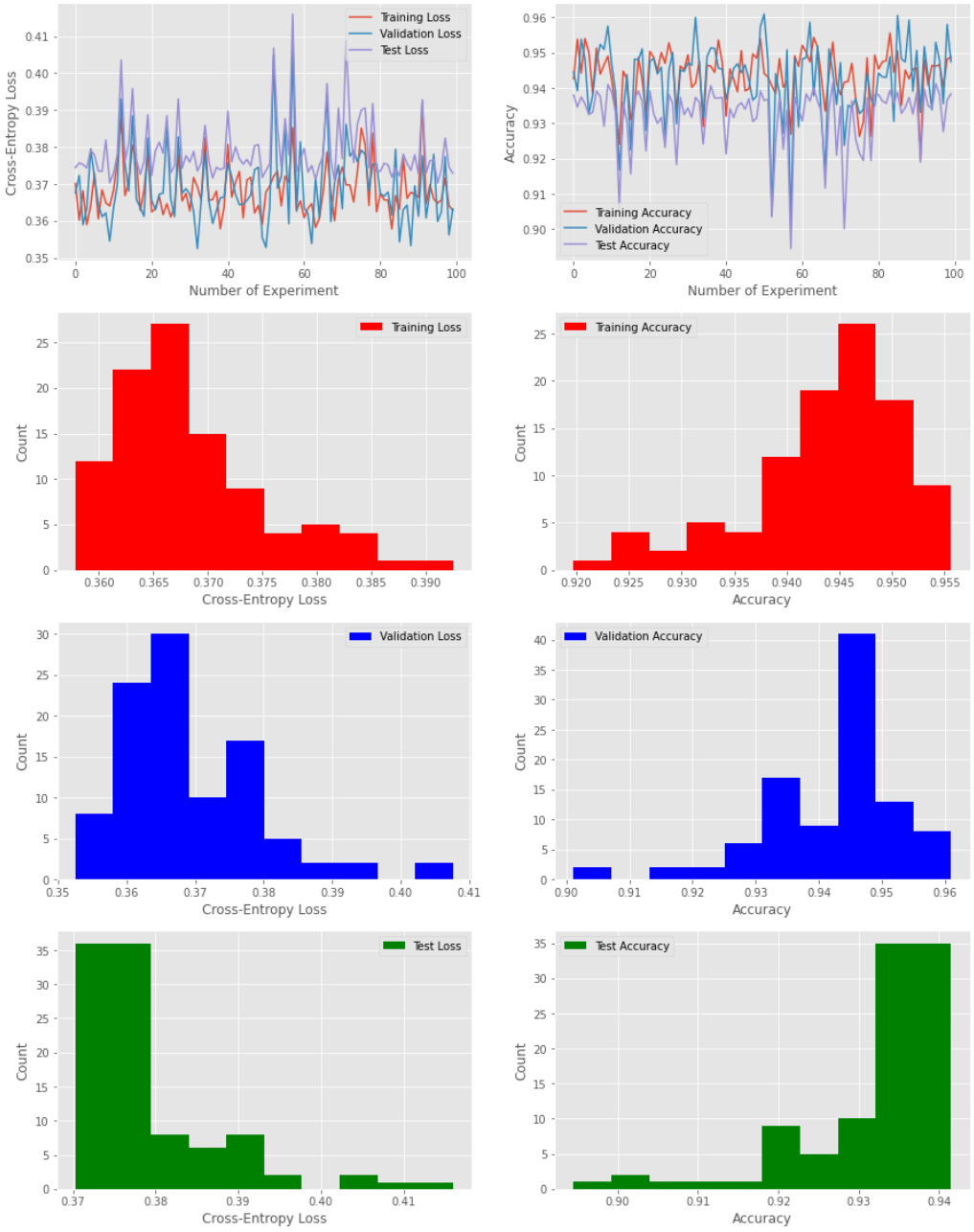

Figure 4.7: Simulation Results Max-Pooling

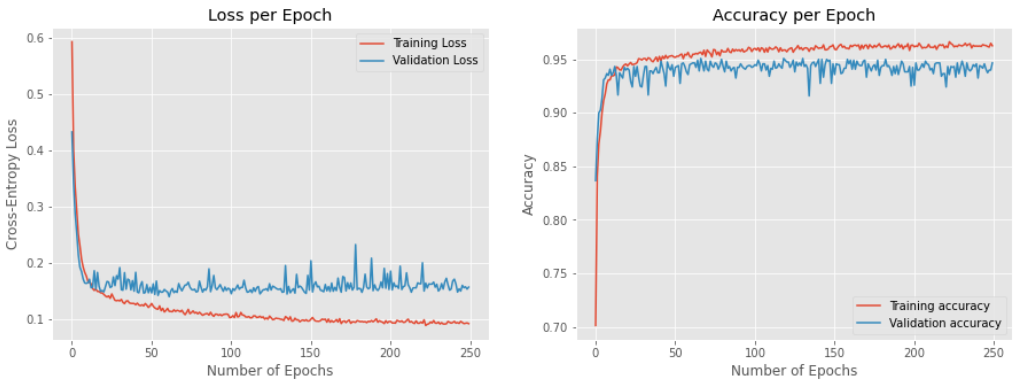

Figure 4.8: ANN Model Fit

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*

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

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* .

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

Erdös, P. & Rényi, A. (2011), On the evolution of random graphs, *in* 'The structure and dynamics of networks', Princeton University Press, pp. 38–82.
URL: *https://doi.org/10.1515/9781400841356.38*

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*

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.

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*

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*