Master's Thesis

Gaining Customer Insights with Machine Learning on Graphs

University of Basel
Faculty of Business and Economics

Degree Program:
Master of Science (MSc) in Business and Economics

Major:
Quantitative Methods

Author:
Michael von Siebenthal

Supervisor:
Prof. Dr. Dietmar Maringer

Submission Date:
July 12, 2021

Abstract

This master's thesis investigates graph machine learning for the purpose of gaining customer insights. Given that customer data is most commonly only available as standard cross-sectional data, semi-synthetic graph generation is investigated. Specifically it is assessed to what extent semi-synthetic graphs can be utilized for graph machine learning. To assess the viability and competitiveness of this approach, different graph machine learning models are tested. In addition, the graph machine learning results are compared to the results of common standard machine learning models. The results of the GraphSage model reveals, that using semi-synthetic graphs can indeed be a viable approach. The semi-synthetic graphs can further provide useful results for visualizing relationships within the data. Lastly it is shown, that the semi-synthetic graph could potentially be used for overcoming the difficulties associated with unbalanced label data. Semi-synthetic graphs are shown to be limited by the extent to which they can capture useful graph structures for predicting the label of a given classification task. The uncommon graph properties of semi-synthetic graphs further limit the capabilities of some graph machine learning models. Suggestions for future research is provided and focuses on improving semi-synthetic graph generation, using graphs for cluster analysis and improving graph machine learning models.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ANN artificial neural network

BFS breadth-first search

CNN convolutional neural network

DFS depth-first search

ETF exchange traded fund

GAT graph attention network

GCN graph convolutional network

GIN graph isomorphism network

GNN graph neural network

GPU graphical processing unit

KDD Knowledge Discovery and Data Mining (International Conference)

LSTM long short-term memory

MAG multiplicative attribute graph

NLP natural language processing

OECD Organisation for Economic Co-operation and Development

QDA quadratic discriminant analysis

RNN recurrent neural network

SVM Support Vector Machine

US United States of America

Chapter 1

Introduction

This thesis investigates models for gaining customer insights using graph machine learning. Graph machine learning is a current frontier in machine learning and has many successful applications in many areas such 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). More generally, there are a vast range of applications for graph machine learning in the fields of natural science, social science and many more as shown by the excellent overview given by Zhou et al. (2020). Graphs are especially useful as they allow for the consideration of relationships between observations. Graph machine learning has for that reason become a promising field as it allows for the use of richer data. This thesis will focus on graph machine learning for the purpose of customer classification. In particular, it is investigated to what extent semi-synthetic graphs can be used for graph machine learning. To provide a better overview as to how this topic relates to business & economics related fields such as gaining customer insights, some motivating examples are provided in the following section.

1.1 Relevance to Economics

From a business & economics perspective, graphs are particularly interesting if one wants to model the interactions between institutions. An example for this is shown in an article published by Schweitzer et al. (2009) which created the graph shown in figure 1.1. This graph depicts the interdependencies of international banks. Such graphs can be useful tools for analyzing such interdependencies and to provide important information for making the banking system more robust and resilient.
Another interesting application of graphs is to model social interactions. While there are many different areas of interest which make use of social connections, the focus of this thesis is placed on gaining customer insights. Indeed, this is one of the

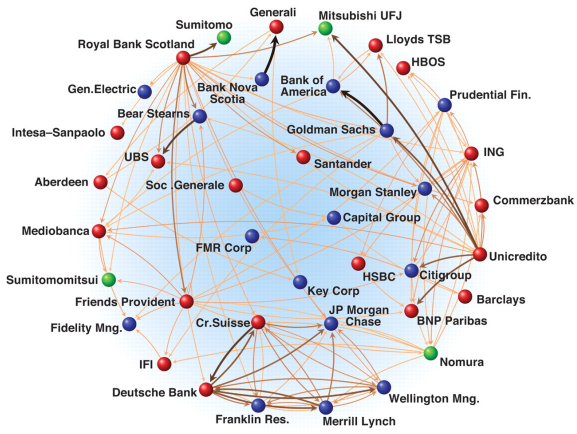

Figure 1.1: Bank Network
(Schweitzer et al. 2009, p. 424)

main areas where social network companies such as Facebook or search providers like Google make their revenue. Those companies mainly generate their revenue by providing customer insights or selling targeted advertising (Facebook 2021, Alphabet 2021). Both Facebook and Google have the advantage, that their businesses naturally capture relational or more generally network data. Most researchers and companies however do not have access to such data. Companies for instance may have access to large amounts of customer data. This data however typically does not contain 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. This makes the collection of network data basically impossible. For that reason, most companies and researchers are limited to working with data such as cross-sectional data that contain no network information. It is important to mention, that there is a lot of network data available online. This network data however typically only contains the network connections. The important feature data such as demographic data, topic specific variables and labels are however typically not available. Without feature data, graphs provide rather limited information for gaining customer insights. This is a data access and data collection problem and is a frustrating reality which also affects this master's thesis. It however motivates the research topic which is presented in section 1.3. First, a general overview of machine learning is given in the following section.

1.2 Overview Machine Learning

This section provides a high-level overview of machine learning and specifies the type of machine learning task used for this thesis. To start, it is important to correctly categorize machine learning. There are many related big topics such as data science,

big data or artificial intelligence and it is often not clear what exactly is meant. An overview of how these different terms can be categorized is shown in figure 1.2.

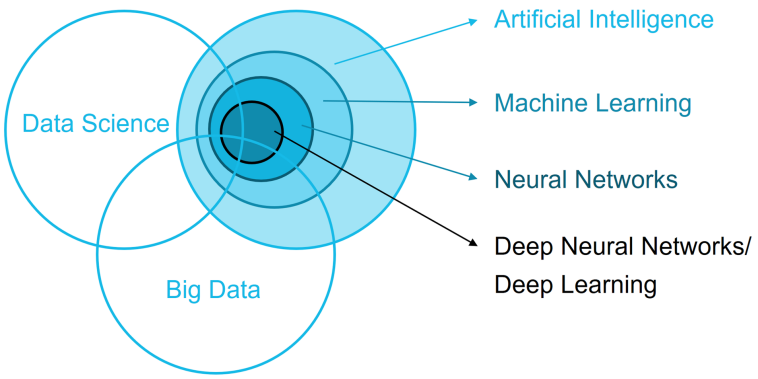

Figure 1.2: Overview Machine Learning
(Frauenhofer Institut 2021)

Figure 1.2 shows well, how these different terms are related with each other. Machine learning in particular is mostly ascribed to the domain of artificial intelligence. It however also has a shared domain with data science and big data. It is thus at the intersection of these three interrelated fields. Machine learning models such as neural networks and deep neural networks are specific models within machine learning and are often referred to separately due to their popularity. In this thesis, differentiating between machine learning and neural networks is not necessary, as the considered machine learning models are used for the same task. Machine learning can be applied for various tasks and is again best presented visually as shown in figure 1.3.

It is shown in figure 1.3, that the main tasks for machine learning involve classification, regression, reinforcement learning, clustering and dimensionality reduction. This thesis focuses on classification tasks. This task is chosen given the available data and because it allows for a nice comparison of different machine learning models. Well known standard machine learning models used for classification tasks include logistic regression (Cramer 2002), naive bayes (Zhang 2004), support vector machines (Platt et al. 1999), random forest classifiers (Breiman 2001), AdaBoost (Freund & Schapire 1997) and artificial neural networks (McCulloch & Pitts 1943). This is an incomplete list of popular machine learning models that can be used for classification tasks. Popular applications of classification tasks include predicting whether a customer is satisfied, whether to grant a mortgage to a client and many more. What the aforementioned machine learning models have in common, is that they can only consider feature data. It is in general not possible to make use of network information for these models.

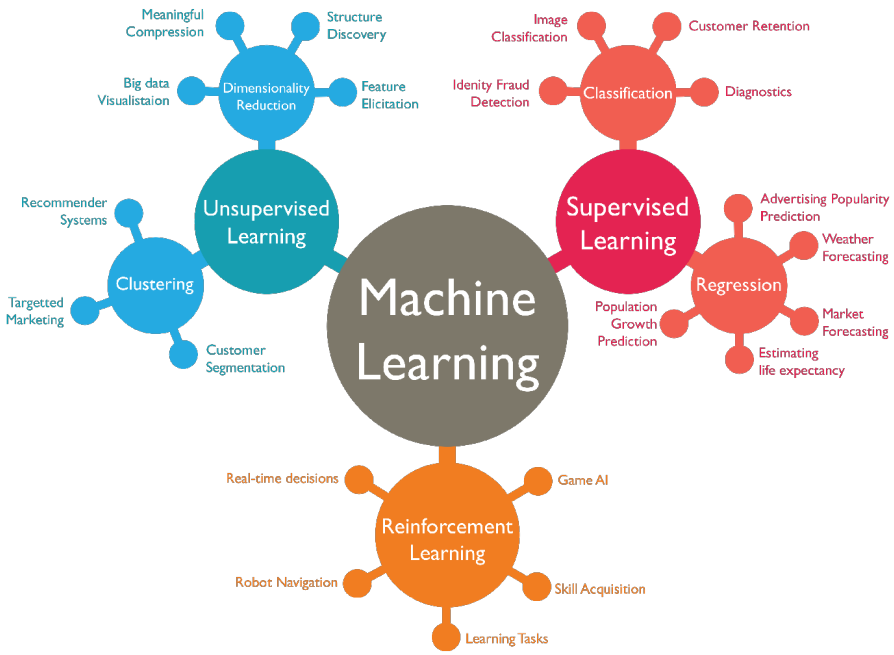

Figure 1.3: Overview Machine Learning Tasks
(Artisan's Asylum 2020)

Graph machine learning methods are different in that they consider both feature data as well as network information. If one wants to categorize graph machine learning within the framework shown in figure 1.2, it is probably best categorized as a special form of neural network. It is however not a deep neural network, as network depth does not necessarily improve the model and can be even counter-productive. Within graph machine learning, there are two main approaches. The first approach focuses on learning vector representations of graphs which are used for downstream machine learning. The downstream machine learning models include the standard models presented previously. Graph representation learning approaches include models such as DeepWalk (Perozzi et al. 2014) and Node2Vec (Grover & Leskovec 2016) among others. The second approach involves the application of graph neural networks for which there exist many different models. These networks include models such as Graph Convolutional Networks (Kipf & Welling 2016), GraphSage (Hamilton et al. 2017) and many more.

The detailed theoretical background for the considered graph machine learning models is provided in chapter 2.

1.3 Research Topic

The difficult access to graphs which also contain features, motivate the search for alternatives. A review of the literature revealed, that a form of synthetic graph gen-

eration could provide a solution to the data access problem. Classic graph generation procedures include the famous Erdös-Rényi graphs (1959), the small-world model by Watts & Strogatz (1998), the well-known model by Barabási & Albert (1999) and more recently Kronecker Graphs by Leskovec et al. (2010). These models are all very instructive regarding the graph generation process and for understanding graph properties. These networks however all have the short-coming, that they do not allow for the assignment of feature data to the nodes/observations in the network. It became clear, that one has to find or develop a model which makes use of existing feature data for the graph generation process. Kim & Leskovec (2012) developed the multiplicative attribute graph (MAG) model. This model makes use of randomly generated feature data which is referred to as attribute data by the authors. The model is shown to be capable of generating random graphs which can adhere to observed real world network properties. An analysis of the MAG model reveals, that it could also be a useful model for creating semi-synthetic graphs using existing feature data. For that reason, the MAG model is selected for generating semi-synthetic graphs. This model is introduced in detail in section 2.3.

More recently, researchers have focused their attention to generative graph models. These models create graphs with features using real graphs as a training input. Examples for such models are graph recurrent neural network (RNN) (You et al. 2018) and deep generative graph models (Li et al. 2018). These are very fascinating models which can be used to recreate or scale graph data. For the purpose of this thesis, these models are not purposeful, as it requires an existing graph with features to be available. Such a graph is unfortunately not available for this thesis. Nevertheless, this is an interesting current topic for graph generation which was considered.

The access problem to graphs which include feature data might be resolved using the MAG model as previously mentioned. This model is interesting as most researchers and companies have access to large amounts of feature data such as customer databases or survey data. It would be of great benefit, if the available feature data could be used for generating semi-synthetic graphs. The MAG model provides a solution for generating such graphs. Semi-synthetic graph refers to the fact, that the graph is generated using real feature data. Fully synthetic graphs on the other hand are generated exclusively using artificial data. The goal of the semi-synthetic graph is to generate connections between the observations in a given dataset. These connections should enhance the information present in the dataset. Graph machine learning should then be capable of exploiting these additional connections and hopefully provide competitive if not superior results compared to standard machine learning models. This thesis therefore investigates to what extent semi-synthetic

graphs can be used for graph machine learning. Of course, semi-synthetic graphs are unlikely to be capable of fully substituting the performance of real graphs for machine learning. Nevertheless, semi-synthetic graphs provide additional and hopefully useful network information. To assess these hypotheses, the results using graph machine learning are compared to standard machine learning models. To close this section, the research question as well as the hypotheses of this thesis are presented formally as follows.

Research Question
To what extent are semi-synthetic graphs based on real feature data useful for a classification task using machine learning?

Hypotheses
H1: Graph machine learning using semi-synthetic graphs provide superior results compared to the results of standard machine learning for a given classification task.

H2: Graph machine learning using semi-synthetic graphs is a competitive strategy compared to the results of standard machine learning for a given classification task.

The required theoretical background for this thesis which includes graph machine learning, graph generation and graph theory in general is provided in chapter 2.

Chapter 2

Theory

This chapter covers the required theoretical background and consists of the following main parts:

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

2.1 Graph Theory

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

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

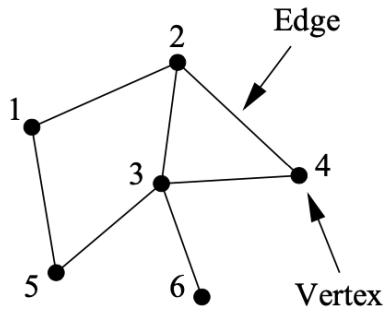

Figure 2.1: Example of a Graph
(Newman 2010, p. 111)

are often referred to as nodes and the terms are used interchangeably. Edges refer to the connections between the vertices. Edges are often also referred to as links and the terms are 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 themselves. This can be considered as a form of feedback loop of a node on to itself. Lastly, multi-edges refer to direct node connections with multiple paths. For notation purposes, graphs are typically defined as follows:

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

G denotes the graph. 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. These aspects are however not relevant for this thesis. For additional information regarding the adjacency matrix, the book "Networks: An Introduction" by Mark Newman (2010) is highly recommended.

Degree Measures

An important measure for graphs are the degrees denoted by k of the vertices. Degrees refer to the number of edges connected to a vertex. The degrees of vertex 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, one can therefore write for an undirected graph with m edges (Newman 2010, p. 133):

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

As a statistical measure, the mean degree c of a vertex is defined as follows (Newman 2010, p. 134):

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

To calculate the density of a graph, it should first be noted, 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 thus 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 correspond to the simplest form of a centrality measure. The issue with this measure is, that the every neighbor of vertex i is valued the same. This is a problem, as not all neighbors are of equal importance due to the:

1. Number of neighbors
2. Importance of neighbors
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 this thesis only considers simple undirected graphs, eigenvector centrality will suffice.

Eigenvector centrality gives all vertices a score which is proportional to the sum of the scores of the vertices 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. One can thus write (Newman 2010, p. 169):

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

In matrix notation:

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

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

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

$x(0)$ denotes the linear combination of (Newman 2010, p. 170):

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

v_i corresponds to the eigenvectors of the adjacency matrix A and c_i corresponds to an appropriately chosen constant. Therefore one 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 equation 2.12, k_i correspond to the eigenvalues of the adjacency matrix A. 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 in terms of 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 of a vertex to the other vertices in the graph. This centrality measure is defined as follows (Newman 2010, p. 182):

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

For this measure, central vertices exhibit high closeness centrality and are therefore more closely 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 equation 2.16, 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 centralities, it is often standardized by the number of connected vertex pairs s and t denoted as η^2. The betweenness centrality can therefore be rewritten as (Newman 2010, p.190):

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

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

2.2 Machine Learning on Graphs

Graph structures are special in that the datapoints 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 via a mutual friend. This aspect is unique to network data and provides interesting additional information as well as added complexity. 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 that reason, graphs cannot be directly plotted as scatter plots since standard distance measures cannot be applied. The graph machine learning methods used for this thesis can be categorized into the following two categories:

1. Graph Representation Learning
2. Graph Neural Networks

Graph representation learning refers to models which generate node embeddings of a given graph. Specifically, this approach creates vector representations of the nodes in a graph given a specified similarity measure. For these vector representations distance measures such as Euclidean distances can be measured. The node embeddings can thus be used for downstream machine learning using standard methods. A graph neural network (GNN) can also generate node embeddings as well as directly

apply graphs to a given machine learning task such as classification. The capability of directly applying graphs for machine learning tasks, makes graph neural networks especially promising.

In the following subsections, the theory for graph representation learning and graph neural networks is introduced.

2.2.1 Graph Representation Learning

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

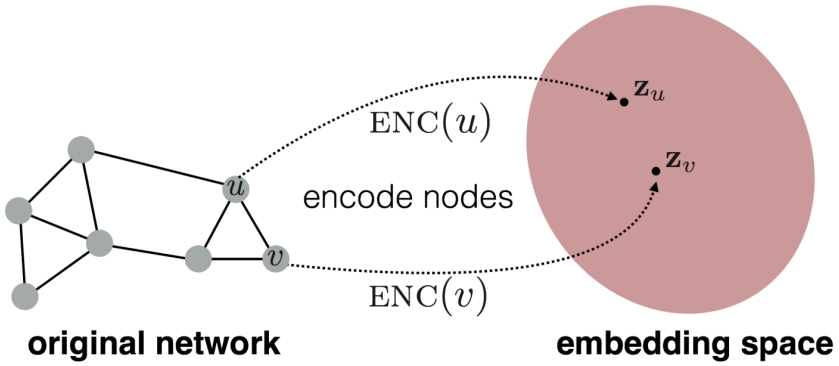

Figure 2.2: Network Embedding
Leskovec (2021)

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

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

The dot product of the two node embedding vectors shown in equation 2.18 should thus approximately equal the similarity of the corresponding nodes in the graph. There are different approaches for defining node similarity. Graph factorization was introduced as an early solution (Ahmed et al. 2013). More recent and successful approaches include methods which make use of random walks. In the context of random walks, similarity is defined as (Leskovec 2021):

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

The models DeepWalk (Perozzi et al. 2014) and its generalization Node2Vec (Grover & Leskovec 2016) successfully apply the similarity measure shown in equation 2.19. Another noteworthy model called LINE (Tang et al. 2015) also makes use of random walk co-occurrences as its similarity measure. In order to remain focused, only DeepWalk and Node2Vec are considered for this thesis. These two models are well suited for the given task and are among the most popular graph representation learning methods.

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

In NLP words are one-hot encoded as inputs for the Skip-Gram model which learns vector representations of the input words. The aim of the Skip-Gram model is then to predict the context of the input word by predicting the neighboring words in a sentence. A basic overview of the Skip-Gram model is provided in figure 2.3.

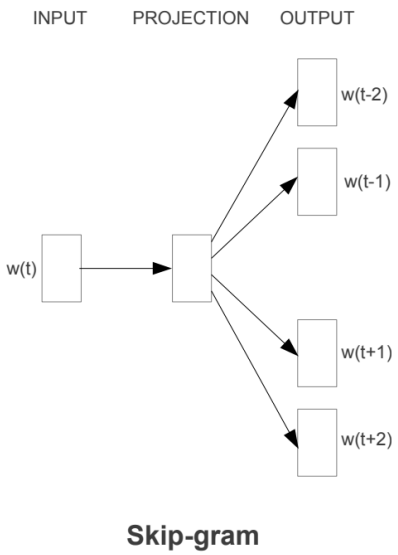

Figure 2.3: Skip-Gram Architecture
(Mikolov et al. 2013a, p. 5)

The basic layout shown in figure 2.3 depicts the high-level procedure of the Skip-

Gram model. To make this model more specific, the input corresponds to the one-hot encoded vector at row t of the input matrix W with dimensions $T \times T$, where every row corresponds to a one-hot encoded word. $W(t)$ is linearly passed to the projection, which involves calculating the dot product of $W(t)$ with the weight matrix Φ which has dimensions $T \times D$. D refers to the number of dimensions which are to be included in the projection vector h and is a hyper parameter. The projection vector h is then linearly passed again with another weight matrix Ψ which has dimensions $D \times T$. This creates the output vector u which is then used to predict the correct context word c from the vocabulary of C number of context words. To do so, the training target is set to maximize the average log probability of the correct context word for every input word w_t. More formally (Mikolov et al. 2013b, p. 2):

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

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

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

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

The same principle shown in the Skip-Gram model can be applied to graphs in a modified version. First, nodes in a graph can be one-hot encoded the same way

as words. This means, that nodes can be used as input data in a similar fashion as words. Based on this idea, DeepWalk by Perozzi et al. (2014) achieved a big breakthrough for graph representation learning. The DeepWalk algorithm builds on top of the Skip-Gram model and uses fixed-length random walks for learning the node embeddings. To provide a better overview of the DeepWalk algorithm, the pseudo-code is presented in algorithm 1 & 2 (Perozzi et al. 2014, p. 704).

Algorithm 1: DeepWalk(G, w, d, γ, t)

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

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

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

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

This is in principle the model which will be used to find the node embeddings of a graph. For the application, the Node2Vec algorithm by Grover & Leskovec (2016)

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

The resulting node embeddings can then be used for downstream machine learning tasks using standard models. 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. The node embeddings correspond to the features used for downstream machine learning tasks. The features were thus learned automatically using the DeepWalk or Node2Vec algorithm. This approach directly takes care of the otherwise at times tedious feature selection process. With this approach, only the number of features need to be defined for feature selection. This is a big advantage and can save a lot of time when working with graphs.

2.2.2 Graph Neural Networks

This section provides an overview of the theory for graph neural networks. Within the family of GNNs, there are a myriad of different models available and every few months new models are published. GNNs are currently very popular and benefit from a large research output. This thesis will focus on two popular and established GNN approaches which are:

1. Graph Convolutional Networks
2. GraphSage

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

- $G(V, E)$ is a graph with a set of vertices and edge connections
- V is a set of vertices
- A is the adjacency matrix of graph G
- $X \in \mathbb{R}^{|V| \times F}$ is a matrix containing the node features
- v is a node $\in V$ and $\mathcal{N}(v)$ is the set of neighbors of node v

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

Modern GNNs have overcome this problem by drawing inspiration from convolutional neural network (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, GNNs 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. An illustration of this mechanism is shown in figure 2.4.

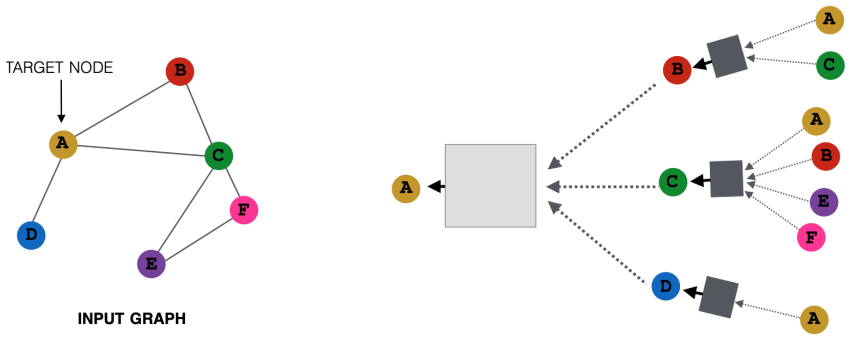

Figure 2.4: GNN Structure
Leskovec (2021)

The GNN structure outlined in figure 2.4 shows an example of a 2-hop or 2 layer GNN. The 1-hop convolutional layer considers the neighboring nodes of the target

node A. The 2-hop layer considers the neighbors of node A's neighbors. Note, that the target node A is included as an input node in the 2-hop layer. This is reasonable as node A itself is also a neighboring node to its neighbors. Taking the example shown in figure 2.4, the challenge for the GNN is to find node embeddings based on local network neighborhoods (Leskovec 2021). The node embeddings at layer 0 correspond to the features of the input nodes where $X = H^{(0)}$. A typical procedure for a GNN model is outlined in the pseudo-code shown in algorithm 3 (Hamilton et al. 2017, Leskovec 2021, You et al. 2020).

Algorithm 3: Typical GNN Algorithm for Model Training

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

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

Algorithm 3 is not meant to be considered as a complete overview and should be rather regarded as an example of a typical GNN structure. In addition, one should split the data into training- and validation sets to ensure a good model fit. GNNs are flexible in that a myriad of modifications can be added to the GNN layers similar to the possibilities of a CNN or a artificial neural network (ANN). The defining features of different GNN models 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) as a reference. 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

the target node A. The output of the final GNN layer corresponds to the node embedding z_A of the target node A. This should appear familiar when comparing this approach to the graph representation learning method outlined in the previous section. GNNs can indeed be used for unsupervised learning tasks such as learning node embeddings. Good examples for generating node embeddings are shown in the articles regarding Graph Convolutional Networks by Kipf & Welling (2016) and GraphSage by Hamilton et al. (2017). As shown in algorithm 3, GNNs can directly be applied for machine learning tasks such as customer classification. This is where GNNs are especially powerful and differ to the Graph Representation Learning algorithms outlined in the previous section. GNNs are flexible tools and can be used in various settings such as graph representation learning, clustering, classification and link prediction tasks among others (Zhou et al. 2020).

Having introduced the general functionality of GNNs, the models graph convolutional network (GCN) and GraphSage are introduced in detail in the following two sections.

2.2.2.1 Graph Convolutional Networks

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

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

The variables in equation 2.22 are defined as follows:

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

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

- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ 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 2.22 is expanded in the following equation for one GCN layer (Dubois 2019):

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

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

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

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

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

2.2.2.2 GraphSage

This section introduces GraphSage by Hamilton et al. (2017) which can be thought of as the inductive counterpart of the GCN presented in the previous section. Inductive refers to the capability of not only performing machine learning tasks on the graph used for training but to apply the trained GNN model to new and 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 GNNs in a practical setting. GraphSage achieves this by applying different aggregation strategies and sampling the neighborhood $\mathcal{N}(v)$. Specifically, the GraphSage model only considers a fixed number of uniformly random sampled neighbors from $\mathcal{N}(v)$ with depth K. A graphical example of this procedure is shown in figure 2.5:

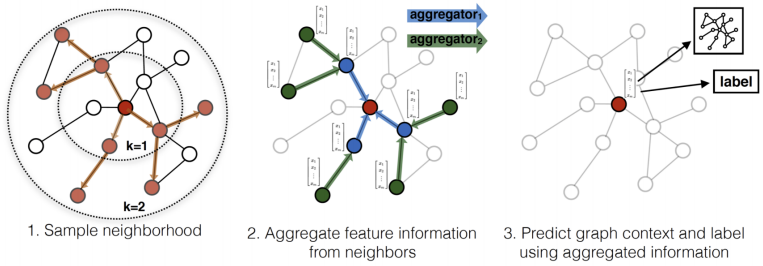

Figure 2.5: GraphSage Sampling
(Hamilton et al. 2017, p. 2)

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

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

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 $W^k, \forall k \in \{1, \ldots, K\}$;
non-linearity σ;
differentiable aggregator functions $AGGREGATE_k, \forall k \in \{1, \ldots, K\}$;
neighborhood sampling functions, $\mathcal{N}_k : v \rightarrow 2^v, \forall k \in \{1, \ldots, K\}$

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

1 $\mathcal{B}^K \in \mathcal{B}$
2 **for** $k = K \ldots 1$ **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** $k = 1 \ldots K$ **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 $z_v \leftarrow h^K_u, \forall u \in \mathcal{B}$

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

1. Mean aggregation
2. Max-pooling
3. Long short-term memory (LSTM) 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 procedure. The difference to GCN is that mean aggregation does not rely on the full graph Laplacian and makes use of a slightly different normalization approach. The aggregation process differs to the one shown in algorithm 4 and replaces the procedures in line 9 and 10 with (Hamilton et al. 2017, p. 5):

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

Max-Pooling Aggregation

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

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

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

LSTM Aggregation

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

2.3 Graph Generation

This section introduces the multiplicative attribute graph model by Kim & Leskovec (2012). This model is used to generate semi-synthetic graphs from feature data as mentioned in the introduction. Originally, the MAG model was introduced for the purpose of generating realistic graphs from feature data and to show that the resulting graph can obey properties of real-world networks. To show this, Kim & Leskovec generated random feature data for which the model parameters were set in such a manner, that the resulting graph adheres to a set of real network properties. These network properties include the emergence of a giant connected component and a power-law or log-normal degree distribution among others (Kim & Leskovec 2012, p. 113). While the creation of a graph which follows real-world network properties would be desirable, it is not the primary target for this thesis. The main goal of the semi-synthetic graph generation is to create a graph that provides useful additional information, which can be exploited using graph machine learning. The MAG model is flexible in this sense, that it can create graphs which are not constrained to adhere to a specific set of network properties. Lastly, Kim & Leskovec (2012, p. 138-139) present model parameters which generate graphs that follow real-world network properties. These model parameters can however not be adopted for the task at hand. The parameters were created for a somewhat simpler task which "only"

involved randomly generated feature data. These randomly generated features do not correspond to any specific features. For that reason, the model parameters can be chosen freely such that the generated network follows real world network properties. The aim of Kim & Leskovec was primarily to create random graphs that can adhere to said network properties. For this thesis, the same MAG model will applied using real feature data. This makes the task of defining the model parameters more difficult as they cannot be freely assigned. In addition, the generated graph will be used for machine learning which is not an application the authors had considered. In this regard the reason and aim for using the MAG model differs for this thesis compared to the original paper.

The starting point of the MAG model is a matrix of feature data, $X^{N\times F}$, where N refers to the number of observations and F to the number of features. Kim & Leskovec refer to feature data as attributes and they use the terms interchangeably. For this thesis a distinction is made, where features correspond to the full set of feature data and attributes refer to the K number of features used for generating the graph, $G(V, E)$. The attribute data, $B^{N\times K} \subseteq X^{N\times F}$, is therefore a subset of the feature data. As a selection criterion, attribute data must be of such a manner that reasonable link-affinity matrices, Θ_i, can be defined. In addition, attribute data must be discrete. For practical reasons, the cardinality of the attributes should not be too large. Attributes such as age might have to be discretized into 4 or 5 discrete age categories. The link-affinity matrix Θ_i is a matrix containing probabilities which are used to estimate the probability of two observations in the dataset (u, v) to form a connection. More specifically, the probability for a connection is calculated given the attributes a_i of observation u and v where $(a_i(u), a_i(v)) \in B$. For example, a reasonable assumption could be to assume, that people which are of the same age group, are more likely to be similar and thus form a connection compared to people which are not of the same age group. Another example for this could be gender in terms of biological sex which is a classical binary setting for a link-affinity matrix. Examples of binary attribute link-affinity matrices Θ_i are given in figure 2.6.

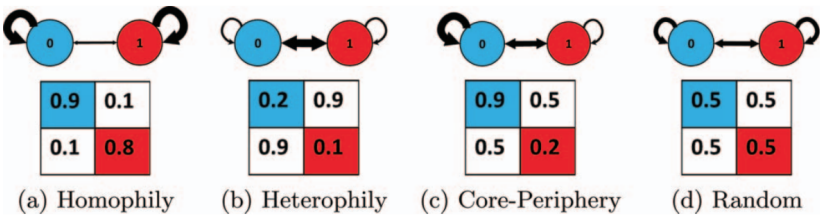

Figure 2.6: Attribute Link-Affinities
(Kim & Leskovec 2012, p. 118)

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

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

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

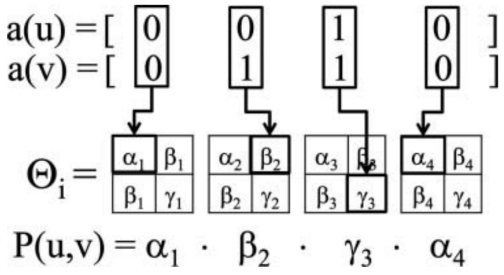

Figure 2.7: Schematic Representation of the Multiplicative Attribute Graphs (MAG) Model

(Kim & Leskovec 2012, p. 120)

The pseudo-code of the MAG model is depicted in algorithm 5 and generates the adjacency matrix A which is then used for constructing the graph $G(V, E)$. The observations of the feature data X which contains the attributes are used as inputs and correspond to the nodes in the resulting graph. More precisely, the order of the

Algorithm 5: Multiplicative Attribute Graph Model

Input: graph node-attribute generation matrix $B^{N\times K}$, where $B \subseteq X^{N\times F}$;
node attribute vector a_i with cardinalities d_i for $i = 1, 2, \ldots, K$;
link affinity matrices $\Theta_i^{d_i \times d_i}$, for $i = 1, 2, \ldots, K$;

Output: adjacency Matrix $A^{N\times N}$ for Graph $G(V, E)$

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

generated adjacency matrix A, corresponds to the ordering of the feature matrix X. Therefore the features can be assigned to the nodes of the generated graph. The procedure outlined in algorithm 5 can be summarized with the following sequential steps:

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

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

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

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

Chapter 3

Data

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

1. Self launched survey
2. Bank Telemarketing dataset
3. US Airline Passenger dataset

The datasets are introduced to the extent that they are successful or useful within the framework of this thesis. In particular, the self launched survey and the Bank Telemarketing dataset showed to be problematic for different reasons. They however provide valuable insights as to when graph machine learning can be successful. These two datasets are therefore only briefly introduced with the focus lying on providing the relevant insights gained from these "failed" datasets. The detailed introduction and analysis of these datasets is therefore skipped. The data and the analyses of these two datasets are provided in the GitHub repository referenced in section 3.1. The airline passenger satisfaction dataset is presented in detail, as good results are achieved using graph machine learning. This dataset is also used for comparing graph machine learning to the standard machine learning models.

Before introducing the datasets, the programming language and the packages used for the analyses are thankfully referenced in the following section. In addition, the GitHub repository containing the datasets and code is referenced.

3.1 Software & Code

The entire master's thesis was evaluated using the Python 3.8.10 programming language (Van Rossum & Drake 2009). In addition, following open-source python packages were thankfully used which are Numpy 1.20.2 (Harris et al. 2020), Matplotlib 3.3.4 (Hunter 2007), NetworkX 2.5.1 (Hagberg et al. 2008), Seaborn 0.11.1 (Waskom 2021), Pandas 1.2.5 (McKinney et al. 2010), Statsmodels 0.12.2 (Seabold & Perktold 2010), Scikit-Learn 0.24.2 (Pedregosa et al. 2011), Tensorflow 2.4.0 (Abadi et al. 2016), Pytorch 1.7.0 (Paszke et al. 2019), deep graph library (dgl) 0.6.1 (Wang et al. 2019), tqdm 4.61.1 (da Costa-Luis et al. 2021) and Node2Vec 0.4.3 (Cohen 2021).

The datasets as well as the Python code used for creating and analyzing the data can be found in a public GitHub repository[1]. The GitHub repository also includes the Python code for the results shown in chapter 4. The repository serves as a general reference for the data and Python code used for this thesis and will not be referenced individually for every analysis and/or result presented. Every folder of the GitHub repository includes a readme file which describes the files present in the folder. In addition, the Python Code is written in Jupyter Notebooks which include descriptions of the code. The implementation of the GraphSage model is partially derived from the tutorial given at the KDD conference (KDD 2020).

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, etc.). The attributes used for creating the MAG graph included mostly demographic data. Additional data was collected which assessed the financial knowledge and behavioral profile of the survey participants by using questions from the financial literacy report of the OECD (2017). The idea was, that demographic data coupled with the financial literacy questions should provide a suitable database for the bank client classification task.

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

[1] GitHub repository: `https://github.com/MichaelvonSiebenthal/MasterThesis.git`

model, the resulting graphs could differ dramatically. This lead to significant performance differences for the different machine learning methods applied to the resulting graphs. Classification accuracies ranged between 40 - 95%. A remedy for this problem could be to assign a fixed probability threshold such as 0.5 in the MAG model. Using a fixed threshold probability, the MAG model would generate deterministic graphs which are always the same. The downside however is, that this makes the graph generation process less realistic. In a homophily setting this would assume that a connection is formed with any node where $P[u, v] > 0.5$. It is unclear without testing, what impact this would have on the graph generation and whether it is positive or negative. It is well understood, that people often form connections with people that appear unlikely from a probabilistic perspective. This consideration would warrant the generation of stochastic graphs. The main priority is however to create graphs which provide additional information that can be exploited via graph machine learning. Having this objective in mind, it warrants further investigation for which the results are presented in section 3.4.2.

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

3.3 Bank Telemarketing Dataset

The Bank Telemarketing dataset first introduced by Moro et al. (2011, 2014) is considered as a banking related back-up dataset for the case that the self made survey does not yield a sufficient number of responses. The Bank Telemarketing dataset is based on a marketing campaign at a Portuguese bank. The dataset includes demographic data, data regarding the bank client's wealth, contact success during previous campaigns among others. The dataset further provides label data which indicates whether a client invested in a short-term deposit after having been contacted by the call center of the bank. The dataset is therefore set-up for a binary classification task. The MAG graph generated from the Bank Telemarketing dataset is shown in figure 3.1.
The red dots in figure 3.1 mark the clients which decided to invest in the short-term deposit and the blue dots did not invest. This figure masks some of the blue nodes due to the figure generation process. The general pattern however is apparent. The red nodes are randomly placed in the network which suggests, that graph machine learning will be of limited use. The graph further shows, that only a relatively

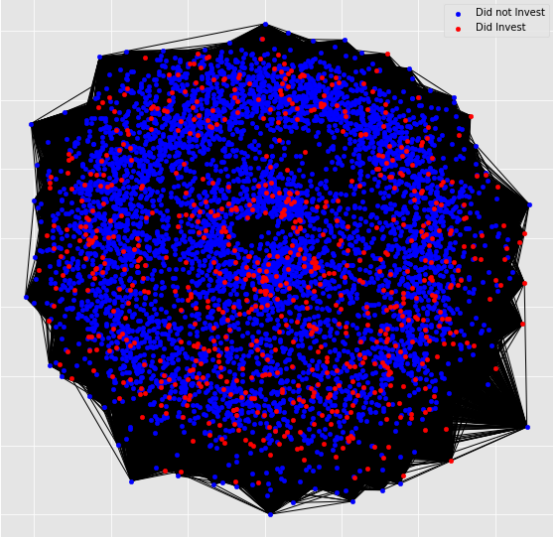

Figure 3.1: MAG graph of bank telemarketing dataset

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

Predicted / **Label**	**Did not invest**	**Invested**
Did not invest	1'000	55
Invested	104	54

Table 3.1: Confusion Matrix Validation Bank Telemarketing Data

The confusion matrix corresponds to an accuracy of approximately 86.89%. The MAG generation process was repeated multiple times for which the GraphSage accuracies ranged between 86 - 90%. Similar results are observed for both graph based methods and standard machine learning methods such as ANNs or Support Vector Machines (SVMs).

Unbalanced datasets are part of a larger and common problem in machine learning. Possible remedies might include using loss functions which penalize false classifications harsher than the standard cross-entropy loss function used for the GNNs.

Alternatively, one could also reduce the dataset by dropping observations so that the remaining dataset is balanced. This approach has its own problems as dropping a large number of observations discards a lot of potentially valuable information. It could also put in question the external validity of the model. These comments point to a separate field of research and could be interesting for a future project.

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 is not possible. In order to check, whether network structure could indeed remedy the unbalanced label problem, the label of the bank telemarketing dataset was used as an additional attribute for the MAG model. The label is normally not included as an attribute for the MAG model, as the label data is usually unknown outside of the training dataset. The link-affinity probabilities for the label is set as follows:

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

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

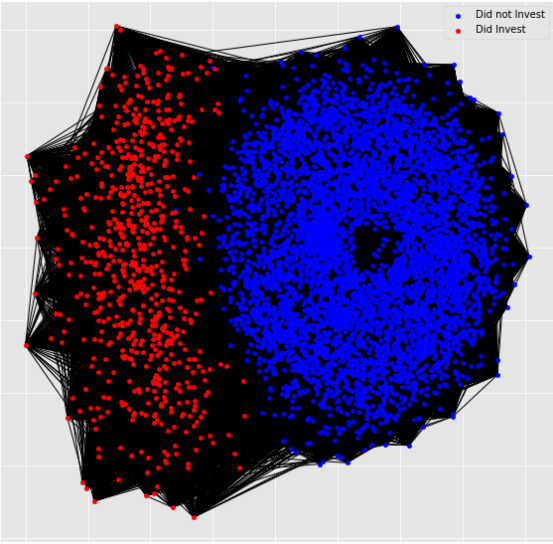

Figure 3.2: Biased MAG graph of bank telemarketing dataset

The nodes shown in figure 3.2 are now nicely clustered according to their label. The GNN method GraphSage achieved an accuracy of over 95% for this graph using otherwise identical feature data and model specifications as before. This is of course a form of cheating, as one cannot assume to know the labels of the graph outside of

the training setting. In a real-world application, the model would be trained using label data and then applied to new data which does not contain the label. This approach would however require the label for the graph generation procedure which makes this not a viable approach. Nevertheless, this result shows extremely well when graph machine learning can yield superior results compared to standard machine learning methods. The key lies in generating a graph with a network structure that corresponds to the label. The network structure need not necessarily be as clearly separated as shown in figure 3.2. A graph which contains local neighborhood clusters which correspond to the same label could yield similar good results. In such a setting, one would have to be careful when defining the number of K layers and neighborhood sampling function $\mathcal{N}_k$. A network structure which corresponds to the label could perhaps also be generated without the label to an extent. This would require the attribute data used in the MAG model to be related with the label. The attributes would have to be substitutes for generating a network structure which correspond to the label. Given the available attributes for the bank telemarketing dataset, this is a difficult task. All features in the dataset have very low correlations with the label. The largest correlation, which is a strong outlier, with the label is call duration with a correlation coefficient ≈ 0.4. This value is rather small and a simulation showed, that it could not be used as a single substitute for the label. Selecting the appropriate attributes and defining the link-affinity probabilities is not a trivial task and requires a lot of trial and error. Unfortunately, for this dataset no appropriate attributes and link-affinities were found that yielded the desired result. For that reason, this dataset is also discarded for further analysis. The dataset and the performed analyses can be found in the GitHub repository.

3.4 US Airline Passenger Dataset

The US Airline Passenger dataset is a survey which was conducted in 2015 by J.D. Power (2015). The dataset was retrieved on the website Kaggle (2020) and is well suited for applying graph machine learning following the MAG generation procedure. The dataset focuses on classifying satisfied- and neutral or dissatisfied passengers. The dataset is thus set up for a standard binary classification task. The dataset further proves to be a competitive dataset for standard machine learning models. This makes this dataset a suitable candidate for a fair comparison of graph machine learning vs standard machine learning. This dataset is presented in detail as it is used for the results presented in chapter 4.

An overview of the US Airline Passenger dataset is shown in table 3.2. The correlation heatmap of the dataset is further shown in figure 3.3. The correlation

heatmap reveals, 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 reveal, that the machine learning models performed better if these variables are included in the model. The data shown in table 3.2 and figure 3.3 corresponds to a random sample of 6'000 observations from the training dataset consisting of 103'904 observations. The training graph is created using this sample of 6'000 observations due to computational time considerations.

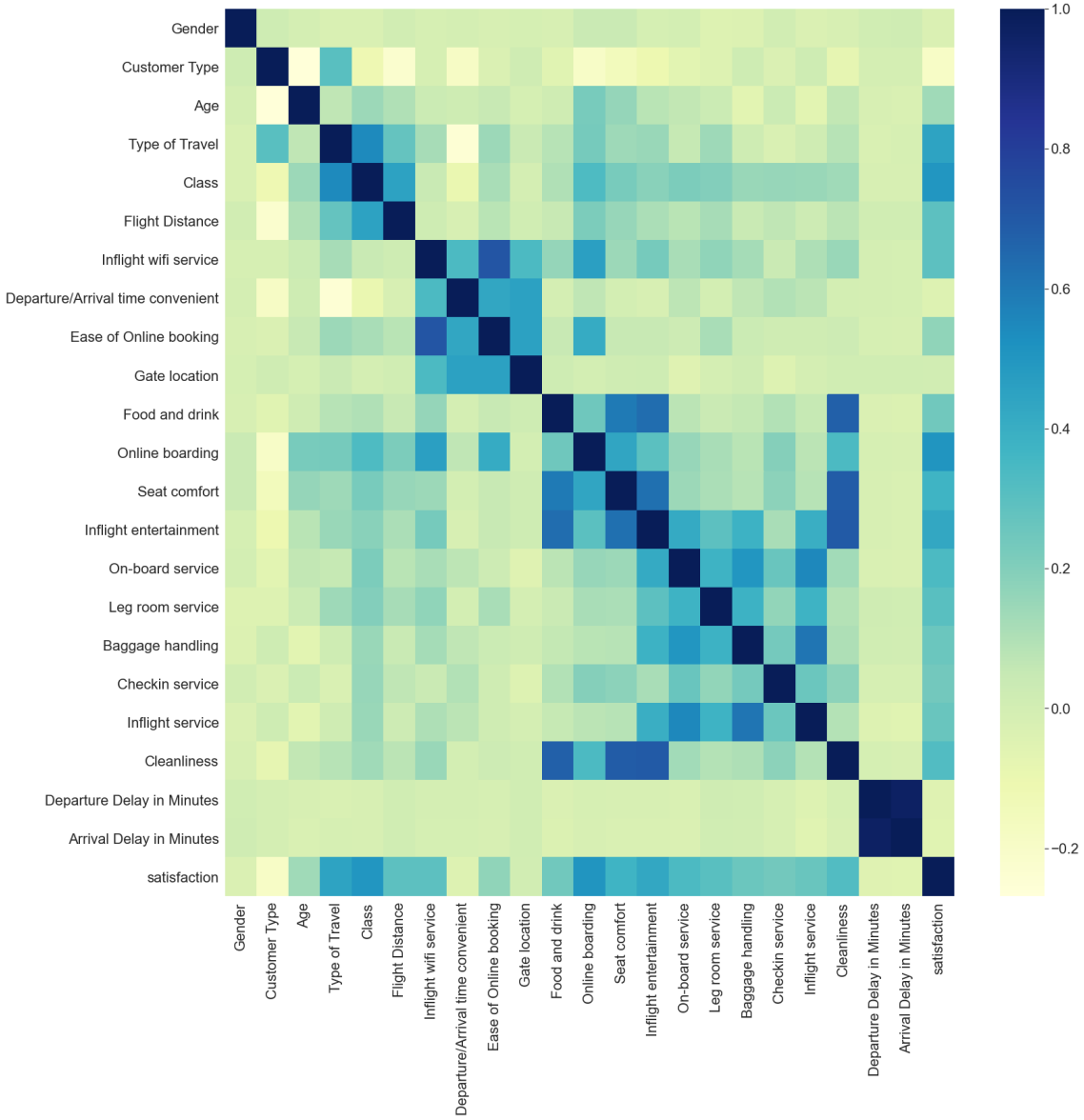

Figure 3.3: Correlation Heatmap of US Airline Passenger Dataset

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

Table 3.2: Airline Dataset Overview

The variables of the dataset are categorized as follows:

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

The categorical variables are dummy coded and the ordinal variable Class is coded as shown in table 3.2. The remaining ordinal variables which measure different satisfaction levels are recorded using a Likert scale ranging from $1-5$. Many passengers did not answer all satisfaction level questions. These responses are recorded with a 0. Therefore, the range of values for the satisfaction level variables range from $0-5$. This encoding works well, as a 0 input in a linear pass function of a (graph) neural network will result in a 0 output value. This type of encoding allows for dealing with missing values for neural networks and other machine learning methods. Lastly, the numerical variables had to be normalized. The popular approach of standardizing the entire dataset was unfortunately not possible. Due to the missing values prevalent in the satisfaction level variables, it must be ensured, that a 0 refers to as a missing response. Note, that a recorded 0 for the numerical variables does not correspond to a missing value. The normalizing function applied is defined as follows:

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

In equation 3.1, x refers to the unnormalized variable and x' refers to resulting normalized variable. a defines the lower bound of the normalization range and b is the upper bound. The numerical variables are normalized to be within the range 1-5. Now, all variables are within a similar range and different scaling should no longer lead to biasing behavior.

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

3.4.1 Graph Generation

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

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

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

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

Attribute 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:

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

The attribute age furhter requires some additional data transformation. Age has a rather large cardinality, which makes it difficult to use for the MAG model. For that reason, age is binned into 4 categories with 0 if age < 26, 1 if $26 \leqslant$ age < 39, 2 if $39 \leqslant$ age < 50 and 3 if age $\geqslant 50$. These bins are chosen according to the interquartile lengths present in the distribution of the variable age.

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

As mentioned in the previous section, a sub-sample of 6'000 observations was retrieved from the training dataset consisting of 103'904 observations. With this random sub-sample and the attributes shown in table 3.3, the adjacency matrix for the resulting graph $G(V, E)$ is generated using algorithm 5. The random sub-sample is retrieved due to the computational cost of running algorithm 5. Simulations which involved creating graphs with different random sub-samples show, that the random sub-samples are representative for the entire dataset. This is further supported when comparing the summary statistics of the sub-samples with the full-dataset. The generated graph is shown in figure 3.4.

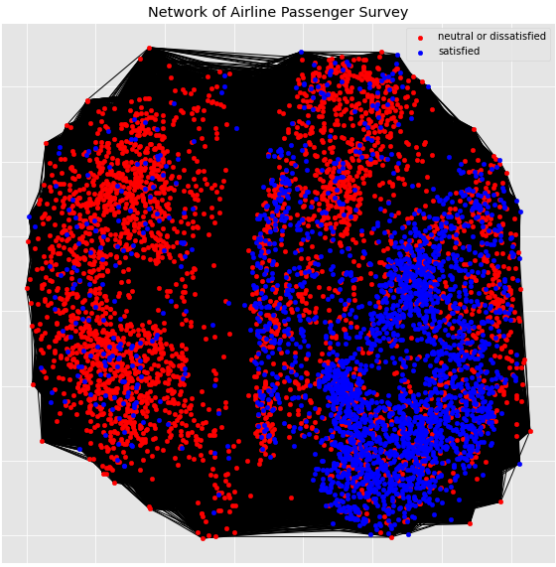

Figure 3.4: Graph of US Airline Passenger Dataset

The network in figure 3.4 shows the emergence of two primary clusters. In addition one can see, that most satisfied airline passengers appear to be grouped together in the right cluster. To gain a deeper understanding of the dynamics involved in the network formation, the nodes of the network are plotted excluding the edges in

figure 3.5.

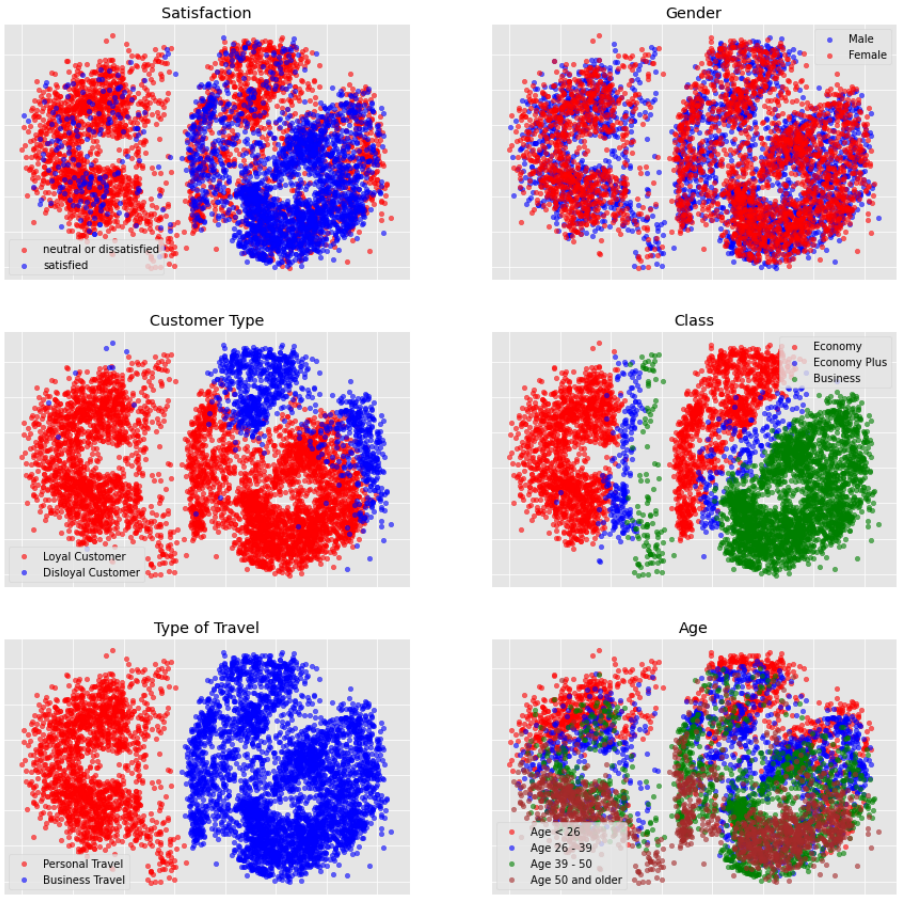

Figure 3.5: Graph Nodes of US Airline Passenger Dataset

Figure 3.5 plots the nodes of the network for the label "Satisfaction" and the 5 attributes used for the graph generation. The transparency of the plot is set to $\alpha = 0.6$ to avoid covering nodes during the graph plotting process. Figure 3.5 reveals interesting associations. First it is shown, that a larger number of people traveling for business purposes are satisfied compared to people traveling for personal reasons. This association becomes clear when comparing the Satisfaction plot with the Type of Travel plot. In relative terms, only approximately 9.8% of passengers traveling for personal reasons are satisfied compared to business travelers with a 57.8% satisfaction rate. Interestingly, passengers traveling for business purposes adhere to somewhat expected characteristics such as:

1. Most business class passengers are satisfied.

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

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

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

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

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

3. Perhaps business travelers fly more frequently than passengers traveling for personal reasons. This could incentivize frequent business travelers to give feedback as they would benefit most from service improvements. Infrequent personal travelers might be less incentivized, as they benefit less from service improvements.

Last but not least, gender does not appear to form any distinguishable clusters. For that reason, it was considered to omit this attribute for the graph generation process. This was tested and the resulting graph was similar to the one shown in figure 3.4. The graph was however more spread out and the neighborhood structures were less clear. In addition, the graph without gender did not perform as well in the subsequent machine learning tasks. Gender appears to provide some useful network information which is why the attribute is kept for the graph generation process.

To provide some more context regarding the graph structure, some graph theoretical metrics are provided. The degree distribution as well as the distributions of the centrality measures: eigenvector centrality, closeness centrality and betweenness centrality are shown in figure 3.6.
The network has a density of approximately 0.0933. This means that 9% of the potential number of connections formed in the network. Nevertheless, when looking

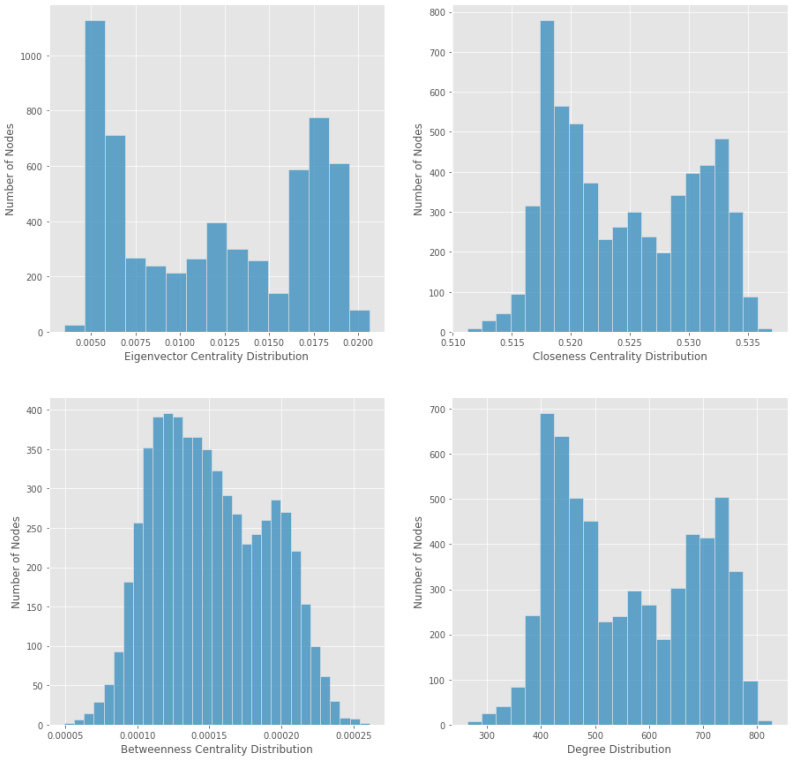

Figure 3.6: Graph Statistics

at the degree distribution histogram we can see, that all nodes have a large number of connections ranging between 263 - 828 with an average of 559.63 connections. The distribution has two modes which likely correspond to the two main clusters shown in figure 3.5. The eigenvector centrality distribution shows, that all nodes have a very low centrality measure and therefore none of the nodes appear to have a large impact in terms of eigenvector centrality. The closeness centrality distribution shows, that all nodes have an average closeness centrality ranging from 0.51 - 0.53. This means that every node is similarly connected and has an average impact for disseminating information across the network. Lastly, the betweenness centrality distribution reveals that there are no bottle-necks through which information flows.

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

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

The power law degree distribution and the 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, it is also expected, that more central nodes in a core-periphery network structure would exhibit some bottle-neck properties for the central nodes. For that reason one would expect central nodes to have a higher betweenness centrality. When looking at the distributions in figure 3.6, this does not correspond to a power law degree distribution. The centrality measures further do not correspond to the centralities frequently observed in real social networks. The graph created with the MAG method does therefore not share the properties observed in real social networks. In order to generate a graph which shares the properties of real networks, one would have to adapt the link-affinity properties to the core-periphery setting as shown in figure 2.6. This was tested and indeed an onion shaped network was created using the same attributes. Forcing this core-periphery structure is however not purposeful for this thesis. Further, the aim of this thesis is not necessarily to create realistic graphs rather than to create useful graphs for graph machine learning. The results in chapter 4 reveal, that the graph is indeed useful for machine learning. For that reason, the generated graph shown in this section is kept for further analysis.

3.4.2 Stochastic vs. Deterministic MAG

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

where the edges are removed. The subgraphs are unfortunately plotted very small, however all subgraphs combined include all 6'000 nodes. The plots are meant to provide a high-level overview of the generated subgraphs.

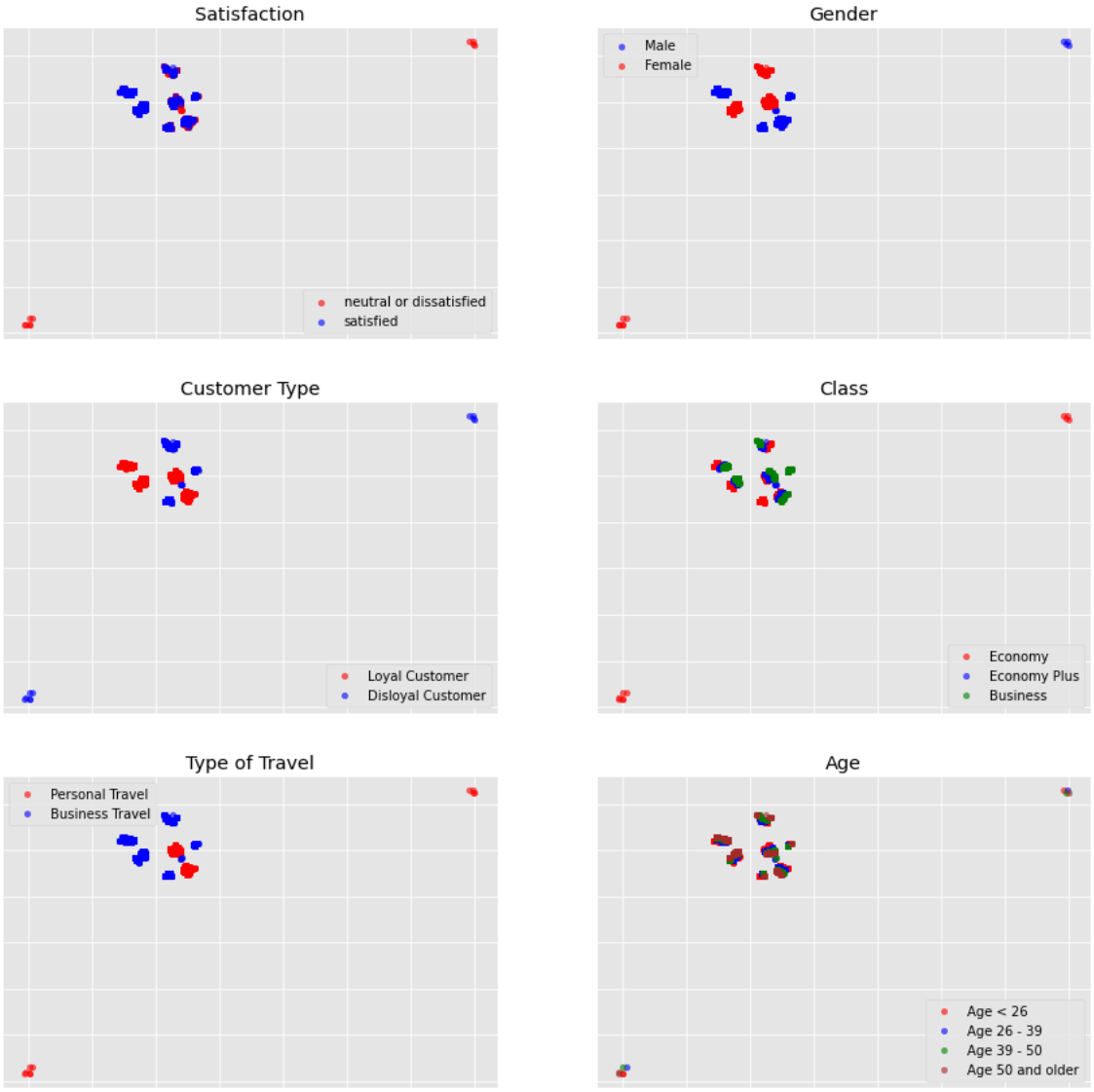

Figure 3.7: Deterministic MAG graph

The deterministic MAG model crates disconnected graphs which form clusters based on node similarities. In terms of performance, the accuracy and loss behavior of both the deterministic- and stochastic generated graphs are virtually identical. This is true for all three datasets considered in this thesis. For the purpose of visualization, stochastic graphs are more useful, as one can identify the different cluster and their relationships to one another 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 subgraphs. These subgraphs could then be used for subsequent machine learning tasks with a cluster specific task in mind. The clusters further could provide infor-

mation as to which clusters tend to be more or less satisfied. This is an area which could be interesting for future research.

Chapter 4

Results

This chapter presents the results for the US Airline Passenger dataset. First, the model specifications and the results using graph representation learning and GNNs are presented. Afterwards, the graph machine learning results are compared to the results using standard machine learning models. For the comparison, the methods logistic regression (Cramer 2002), naive bayes (Zhang 2004), support vector machine (SVM) (Platt et al. 1999, Chang & Lin 2011), random forest classifier (Breiman 2001), AdaBoost classifier (Freund & Schapire 1997, Hastie et al. 2009), quadratic discriminant analysis (QDA) (Tharwat 2016) and artificial neural networks (ANNs) (McCulloch & Pitts 1943) are considered.

4.1 Graph Representation Learning

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

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

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

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

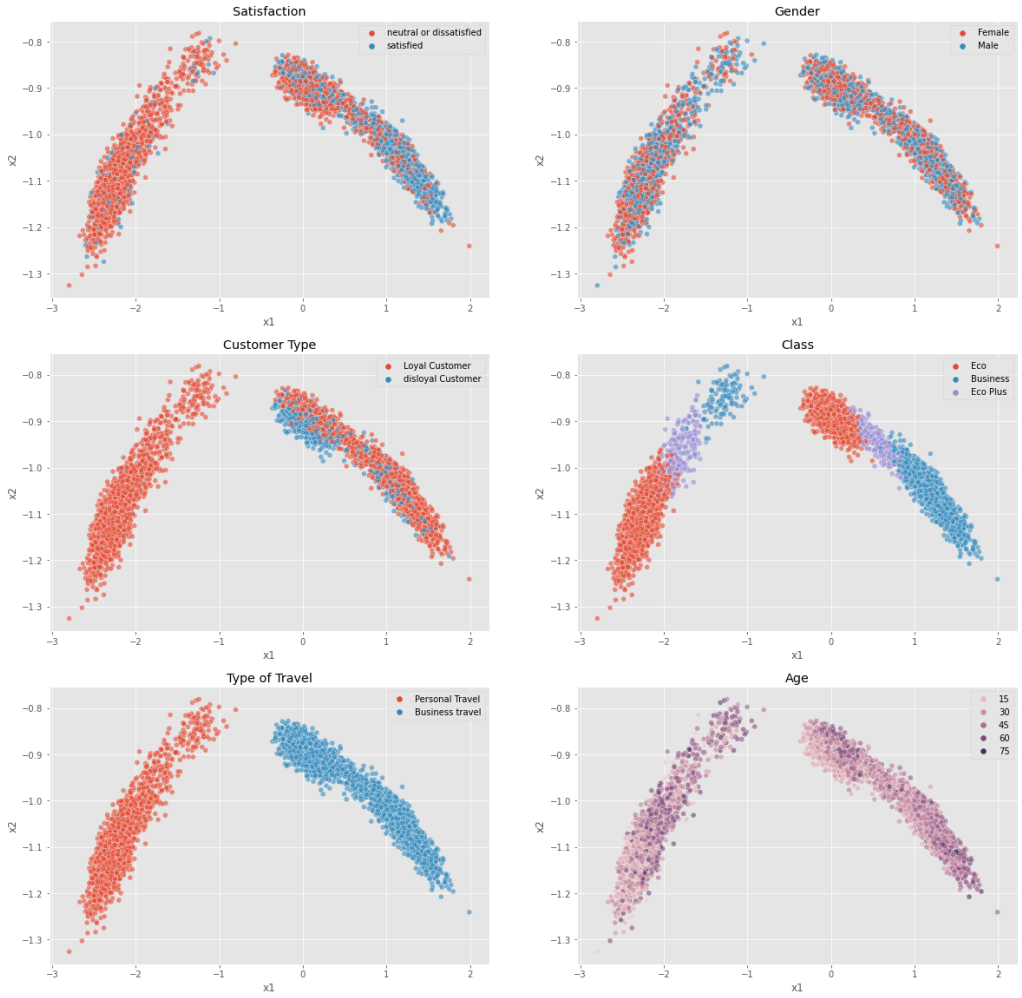

Figure 4.1: Node2Vec Embeddings

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

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

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

Table 4.1: Node2Vec Classification Results

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

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

4.2 Graph Neural Networks

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

methods.

4.2.1 Graph Convolutional Network

The GCN is designed using a similar forward propagation function as the function shown in equation 2.24. The only difference is, that the output layer is activated using the logSoftmax function instead of the Softmax function. The outputs of both activation functions are theoretically identical. The logSoftmax function is however numerically more stable, as it internally makes use of the log-sum-exp trick. A good explanation of this trick is given on the website by Gundersen (2020)[1].

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

The node features X include the 21 explanatory variables of the US Airline Passenger dataset as described in section 3.4. The categorical variables are one-hot encoded, which is why the feature matrix includes 24 variables. The hidden layer size of the convolutional layers is also set to 24 and the output layer is set to 2 for the binary classification task. The training loss is calculated using cross-entropy loss and the model parameters are 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 procedure with the function shown in equation 4.1. In addition, the model parameters are updated using the Adam optimizer instead of standard gradient descent. Different model specifications were tested, for which the chosen specifications perform best. The GCN requires approximately 1'000 epochs to finish training. The resulting loss- and accuracy plots are shown in figure 4.2.

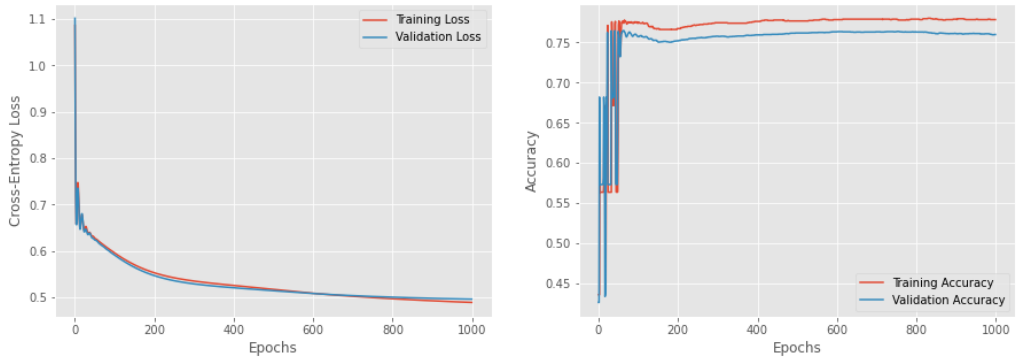

Figure 4.2: GCN Loss- and Accuracy Plots

The confusion matrices and the final accuracies are shown in tables 4.2 & 4.3. 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

[1]Website Gregory Gundersen:
`https://gregorygundersen.com/blog/2020/02/09/log-sum-exp/`

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	776	232
Satisfied	164	616
Accuracy	77.85%	

Table 4.2: Confusion Matrix Training Data GCN

Predicted / Label	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	1'828	587
Satisfied	424	1'372
Accuracy	76.00%	

Table 4.3: Confusion Matrix Validation Data GCN

dataset are used for training and 70% of the data is 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 neighborhood function $\mathcal{N}(v)$, they are just not considered as target nodes for learning and updating the model parameters. The fact, that GCNs cannot be used in an inductive setting is very limiting. Further, the GCN requires 1'000 epochs to 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. GraphSage for comparison is specifically designed with mini-batch training and neighborhood sampling in mind. The results for GraphSage are presented in the following section.

4.2.2 GraphSage

For GraphSage the exact same data input is used as for the GCN. The GraphSage model includes 2 convolutional layers with a hidden layer size of 24 and an output layer size of 2. The model is defined using an adaptation of algorithm 4. Specifically, the hidden layer is activated using the ReLU function with subsequent L2-normalization. The output of the second layer is activated using the logSoftmax function. Normalization is skipped for the final output. The training data 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. The GraphSage model employed is thus an adaptation of the algorithm shown in algorithm 3. The model is trained using a mini-batch size of 50 nodes and the neighborhood function $\mathcal{N}(v)$ randomly samples 10 neighbors at a 2-hop distance from the target node and randomly samples 5 nodes at a 1-hop distance. This corresponds to the steps shown in figure 2.5. To

improve the robustness of the model, a dropout rate of $p = 0.02$ is set. GraphSage is run using the aggregation strategies mean, LSTM and max-pooling. In addition, sum-pooling is added as an additional aggregation strategy as suggested by Xu et al. (2019).

The sum-pooling applied in the GraphSage setting follows the same procedure as max-pooling. The only difference is that element-wise sum-pooling is performed instead of element-wise max-pooling. Xu et al. (2019) show that certain graph structures can confuse aggregation strategies such as mean aggregation or max-pooling. The issue with these aggregation strategies is, that they are not injective functions. Different input nodes (with different graph structures) in a mean- or max aggregation setting can yield an identical output. Given an identical output, the GNN would classify the two identical outputs as the same node representation even though they are the result of different graph structures. This is an area where GNNs can fail to distinguish graph structures. Examples of where sum aggregation succeeds and mean- and/or max aggregation fails is shown in figure 4.3:

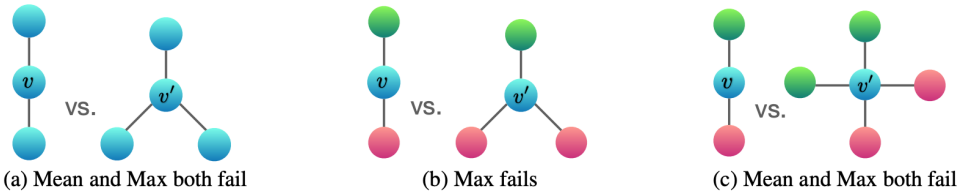

Figure 4.3: Examples of Graph Structures that Mean and Max Aggregators Gail to Distinguish.

(Xu et al. 2019, p. 6)

The node coloring shown in figure 4.3 corresponds to one-hot encoded types of nodes. Further, v and v' should correspond to different node representations, which mean- and/or max aggregation fail to distinguish from each other (hence the same node coloring). Given one-hot encoded nodes with no feature data, sum aggregation is shown to be an injective function (Xu et al. 2019, p. 5). For the US Airline Passenger dataset with its many features, sum aggregation is unfortunately not an injective function. It however sparked the interest in sum aggregation, which is why it is added as an aggregation strategy of interest for the GraphSage model.

The model for each aggregation strategy is 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 test graph which also consists of 6'000 nodes. The test graph is created analogues to the training- and validation graph using a separarte test feature dataset. The results

for all aggregation strategies are presented in the following paragraphs.

Mean Aggregation

In figure 4.4 the training- and validation losses as well as the accuracies for the mean aggregation model is shown.

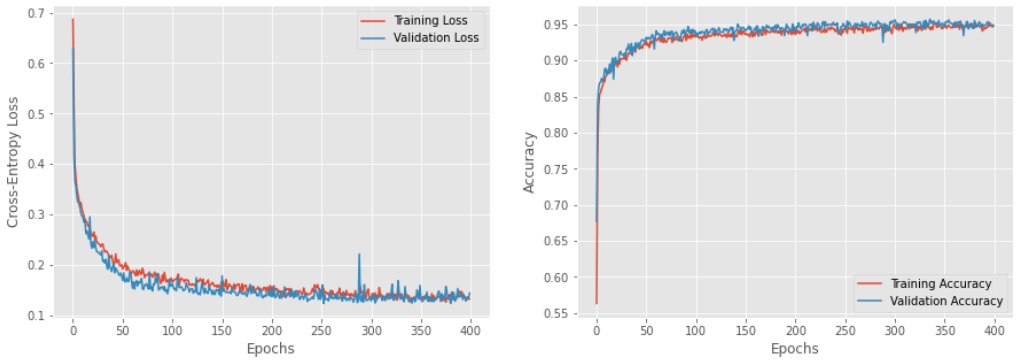

Figure 4.4: Mean Aggregation Loss- and Accuracy Plots

The training accuracy is 94.91% and the validation accuracy is 94.80% after training the model for 400 epochs. The loss- and accuracy plots show a good model fit which remains stable. The model resulted in a test accuracy of 94.08% with the confusion matrix shown in table 4.4:

Label \ Predicted	Neutral or Dissatisfied	Satisfied
Neutral or Dissatisfied	3'278	88
Satisfied	267	2'367
Accuracy	94.08%	

Table 4.4: Test Confusion Matrix Mean Aggregation

LSTM Aggregation

Figure 4.5 shows the training- and validation loss and the accuracy using LSTM aggregation. The model yields very good results for the training- and validation data sets. The LSTM model is however significantly slower for training purposes compared to the other aggregation methods. This is an inconvenient short-coming. The training- and validation accuracy after 400 epochs is 95.75% and 95.14% respectively. Here again, it is shown that the training behavior is relatively good. The model is however prone to over-fit as shown in figure 4.5. Fortunately, the over-fit is small and remains stable. The results for the test graph are shown in table 4.5. The results reveal, that the accuracies of $> 95\%$ for the training- and validation datasets do not transfer to the test graph. While this is to be expected, the decrease in

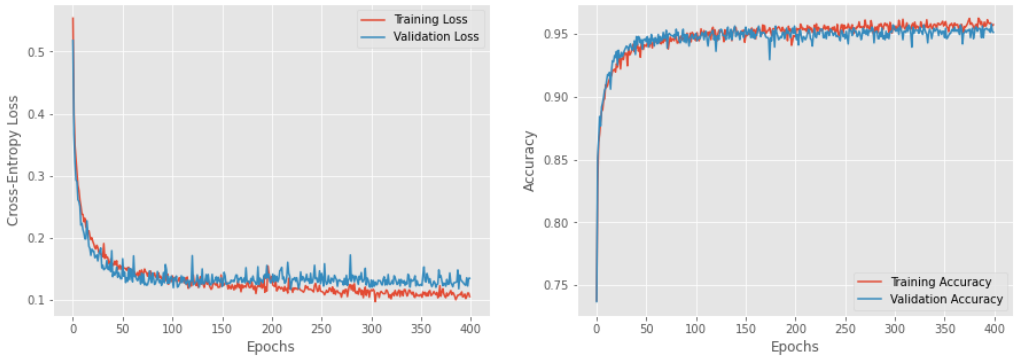

Figure 4.5: LSTM Aggregation Loss- and Accuracy Plots

accuracy for the test graph is most pronounced for LSTM aggregation. This further indicates, that LSTM aggregation is prone to over-fitting.

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

Table 4.5: Test Confusion Matrix LSTM Aggregation

Sum-Pooling Aggregation

Sum-pooling is added for the reasons previously outlined. It is of particular interest to assess, whether sum-pooling can provide some added value compared to the other aggregation strategies. Given the feature data, sum aggregation is unfortunately not injective. For that reason, it is not expected to necessarily dominate the other strategies. Nevertheless, it is of interest to assess, whether sum aggregation is competitive if not superior even in a non-injective setting. The training- and validation results are shown in figure 4.6.

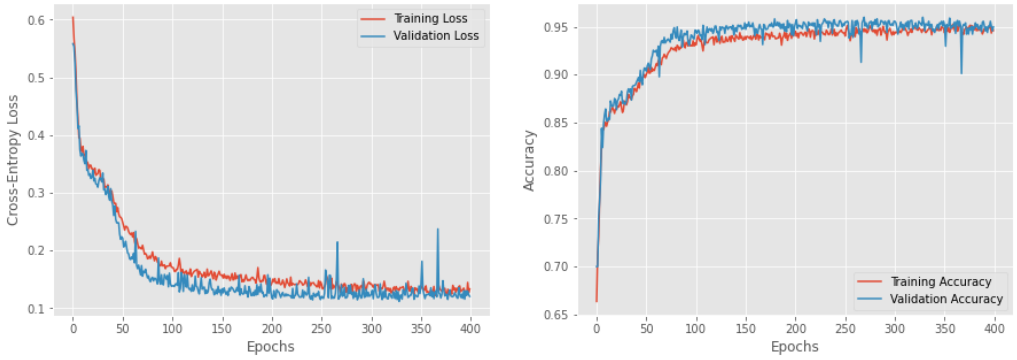

Figure 4.6: Sum-Pooling Aggregation Loss- and Accuracy Plots

The plots show that sum-pooling finishes with a good model fit. The loss curves are however not as smooth compared to the other aggregation strategies. After training for 400 epochs, the training accuracy is at 94.60% and validation accuracy is 94.97%. The results for the test graph are shown in table 4.6. Sum-pooling yields competitive results, it does however not dominate the other aggregation strategies.

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

Table 4.6: Test Confusion Matrix Sum-Pooling

Max-Pooling Aggregation

The training- and validation loss as well as the accuracies of max-pooling are shown in figure 4.7.

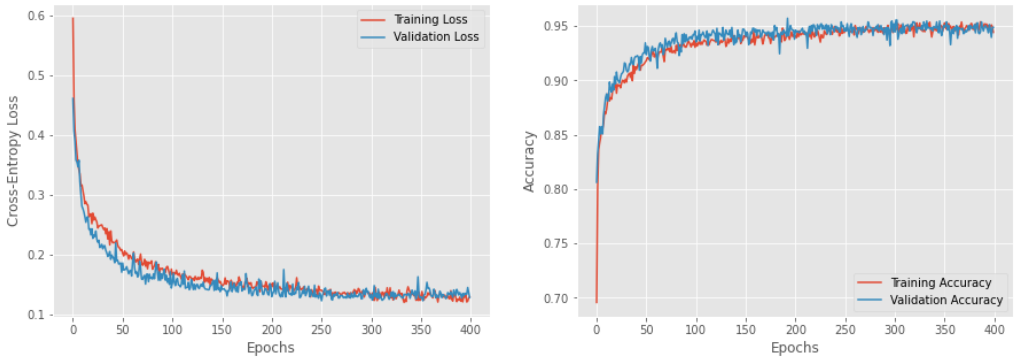

Figure 4.7: Max-Pooling Aggregation Loss- and Accuracy Plots

After 400 epochs, the model arrived at a training accuracy of 94.43% and a validation accuracy of 94.88%. The results for the test graph are shown in table 4.7. Max-pooling yields the best results and corresponds to the recommended method by Hamilton et al. (2017, p. 9).

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

Table 4.7: Test Confusion Matrix Max-Pooling

4.2.3 GraphSage Robustness Simulation

The loss- and accuracy plots using the GraphSage model reveal mostly a relatively good model fit. Repeatedly training the GraphSage models revealed, that there is some non-negligible variation in the performance of the trained models. In particular, depending on how the data is randomly split into 80% training data and 20% validation data, the model results could differ significantly. Simulations reveal, that the random assignment of training- and validation data is not the sole culprit for the observed variation in model performance. The dropout rate of 2% employed in the GraphSage model is just as responsible. Simulations show, that if the dropout rate is set to 0, that the model tends to over-fit significantly. In this setting, the training accuracy approaches 98% while the validation accuracy stagnates around 92-93%. The trained model accordingly does not translate well to the test graph for which the accuracy is approximately 91%. This training behavior is observed regardless of the random train- and validation data set assignment. For that reason, a dropout rate of 2% is set to avoid this over-fitting problem which in turn also yields better test results as shown in section 4.2.2.

Setting a dropout rate introduces an additional obstacle for training the model. While training the model, the dropout rate is applied for forward propagating the training data. The validation data is however forward propagated with no dropout. This is the intended mechanism which can be used to prevent over-fitting. This however makes the model more sensitive to the random assignment of nodes into training- and validation data. Given the structure present in the graph, some nodes are more difficult to classify than others. If the training data includes more difficult nodes in combination with the dropout rate, the outcome can occur where the validation data has a lower loss and a higher accuracy than the training data. The same can be true in reverse, where an over-fit occurs if the training data includes mostly nodes which are more simple to classify than the validation data. Lastly, a good model fit is also frequently observed as shown in the results presented in the previous section.

Due to this observed variation in model fit and performance, a simulation is performed with 100 experiments. Every experiment makes use of the same training & validation graph and test graph. For every experiment, the training- and validation set is randomly assigned using a 80/20 split. Afterwards, the model is trained using 400 epochs and is then applied to the test graph. The loss and accuracy for the training, validation and test data is collected for every experiment. These experiments are conducted to ensure, that the results shown in section 4.2.2 are representative.

The results of the 100 experiments using GraphSage with max-pooling is shown in table 4.8 and figure 4.8.

Metric	**Training Set**	**Validation Set**	**Test Set**
Average Accuracy	94.95%	94.90%	93.79%
	(0.65%)	(0.75%)	(0.41%)
Average Cross-Entropy Loss	0.1276	0.1313	0.1625
	(0.0140)	(0.0170)	(0.0108)

Table 4.8: Simulation Results Max-Pooling

Max-pooling is selected as it corresponds to the recommended aggregation strategy (Hamilton et al. 2017, p. 9). The same simulation using 100 experiments is also performed for sum-pooling. The results are almost identical, where max-pooling is marginally more successful. For the remaining aggregation strategies, only 10 experiments are run due to time considerations as 100 experiments require approximately 15 hours to complete. The required time for the simulation could most definitely be shortened using graphical processing units (GPUs). Unfortunately, a GPU is not available for conducting the simulations. If available, this would be highly recommended to shorten the required time for completing the simulations. The results using only 10 experiments yields similar results, with test accuracies ranging mostly between 93-94%. The simulation results for mean, LSTM and sum aggregation are shown in appendix A.1. The results shown in section 4.2.2 are therefore representative results, even if admittedly on the more positive side.

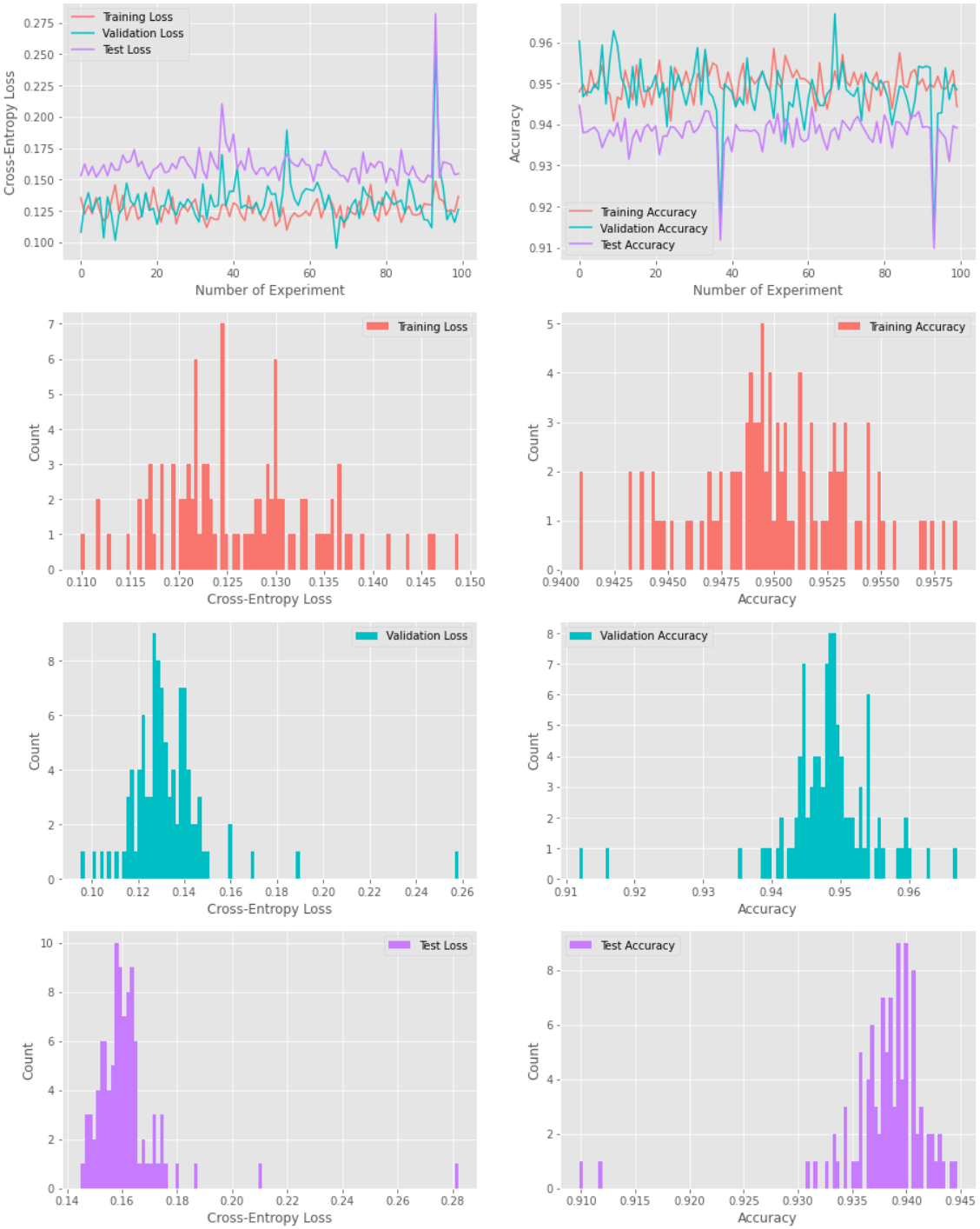

Figure 4.8: Simulation Results Max-Pooling

4.3 Result Comparison

This section compares the results of the three models Node2Vec, GCN and GraphSage. In addition the graph based machine learning methods are compared to the standard machine learning models. More precisely, the same standard machine learning models are used for comparison as for the downstream machine learning task for Graph Representation Learning shown in section 4.1. For the comparison, the standard machine learning models consider the same features as inputs as the GNNs. The comparison is done to assess to what extent, semi-synthetic graphs are useful for graph machine learning. This comparison thus provides the key results for answering the research question and assessing the hypotheses stated in section 1.3. The comparison of the results is shown in table 4.9.

Method	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**
Logistic Regression	87.94%	87.17%	86.77%
Naive Bayes	87.50%	86.17%	85.82%
QDA	70.5%	70.00%	69.23%
AdaBoost	94.06%	92.33%	93.00%
Random Forest	100%	94.42%	94.55%
SVM	94.65%	92.25%	92.73%
ANN	94.86%	95.00%	93.13%
Node2Vec (Logistic Regression)	77.04%	76.16%	57.06%
GCN	77.85%	76.00%	-
GraphSage (Mean Aggregation)	94.91%	94.80%	94.08%
GraphSage (LSTM Aggregation)	95.75%	95.14%	94.23%
GraphSage (Sum-Pooling)	94.60%	94.97%	94.27%
GraphSage (Max-Pooling)	94.43%	94.88%	94.28%

Table 4.9: Result Comparison

The results for the graph machine learning methods shown in table 4.9 correspond to the results shown in sections 4.1, 4.2.1 and 4.2.2. For Graph Representation Learning (Node2Vec), the results using logistic regression for downstream machine learning are shown. This method is selected, as it provides the best overall results in terms of accuracy and model simplicity.

The comparison shown in table 4.9 reveals, that GCN and Node2Vec are not competitive strategies for the US Airline Passenger dataset. GraphSage is shown to be a serious competitor and is the second best method. The Random Forest classifier is consistently the best method regardless of the training- and validation data split and used test data. The ANN is the third best model and is only marginally inferior to the GraphSage models. Similarly as for GNNs, it is important to evaluate the model fit of the ANN to draw a more definitive conclusion. The loss- and accuracy plots of the ANN are shown in figure 4.9.

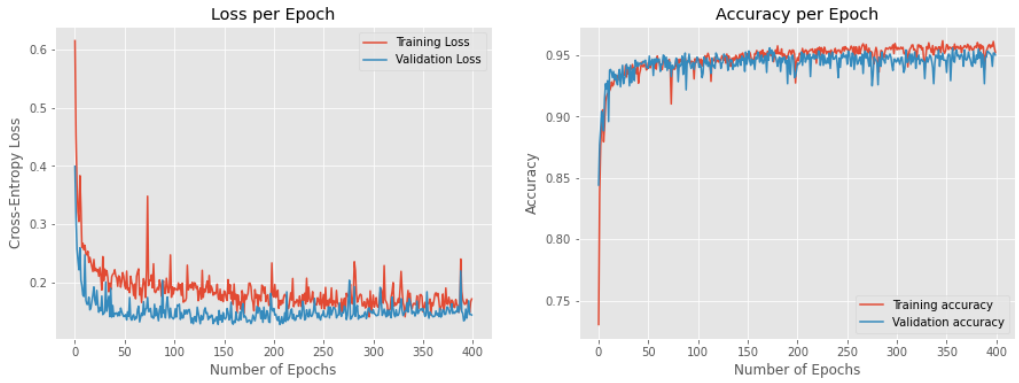

Figure 4.9: ANN Model Fit

The ANN has an input layer size of 24, a hidden layer size of 15 and an output layer with size 2. The output of the first layer is activated using the ReLU function and the output of the second layer is activated using the Softmax function. The loss is calculated using cross-entropy and the model parameters are updated using the Adam optimizer with a learning rate of $\alpha = 0.002$. The dropout rate is set to $p = 0.01$. Without setting the dropout rate, the ANN is prone to significant over-fitting. The presented model settings correspond to the ones which performed best for the ANN. Figure 4.9 shows, that the ANN model starts over-fitting after approximately 200 epochs. When comparing the GraphSage training plots with the ANN, the GraphSage model appears to exhibit preferable training behavior on average.

AdaBoost and SVM also are shown to yield good results. The test accuracies are however not as competitive compared to the previously mentioned methods. Lastly naive bayes, QDA and logistic regression clearly yield inferior results.

Chapter 5

Discussion

The analysis of the data in chapter 3 and the presentation of the machine learning results in chapter 4 provided a myriad of interesting results. These results primarily touch following topics and provide the structure for the discussion:

1. Informativeness of semi-synthetic graphs
2. Appropriateness of semi-synthetic graphs for machine learning
3. Graph machine learning versus standard machine learning

5.1 Informativeness of Semi-Synthetic Graphs

The graph generated for the US Airline Passenger dataset presented in section 3.4 provided interesting results for cluster analysis. The MAG model was successfully used for generating a graph that captured useful neighborhood structures as shown in figure 3.5. Specifically, similar nodes were grouped together based on their similarity given the selected attributes for the MAG model. Especially appealing is that neighborhoods are also consistent within neighborhoods. To illustrate this, one can observe that all nodes are clustered together based on their similarity even within the two main clusters which formed based on the attribute type of travel. This observation is consistent for several additional levels. For instance, nodes form neighborhoods based on age within the cluster of business class passengers which is part of the cluster of passenger which are traveling for business purposes. The only attribute for which no significant clusters emerged is the attribute gender. The reason for this is, that the link-affinity matrix formed a connection with probability of 0.6 for observations with the same gender and 0.4 if they were of the opposite gender. These probabilities were set based on the assumption that observations of the same gender are more similar than the opposite gender. It is however not assumed, that the passenger experience for women would be significantly different to the one

for men. In addition, one needs to be careful when assigning link-affinity probabilities. The data is almost perfectly 50/50 split into male and female observations. If one assigns very low- or high probabilities, this will significantly alter the graph. This is for instance done for the attribute type of travel which split the graph into the two main clusters. This aspect is less of a concern for attributes with multiple categories such as age. Nevertheless, the attribute gender did generate some mild clustering within the network, which could be exploited for graph machine learning. One can observe this mild clustering in the graph, the effect is however not pronounced enough to draw any conclusions from visual observations. The graphs shown in figure 3.5 thus provide very useful insights for better understanding the relationships within the data. This is a welcome bi-product of the MAG model and it is surprising to see how informative the resulting graph is for analyzing the data. In fact, it provides vastly superior insights compared to what a standard scatter plot can reveal given the available feature data. For reference, the pairplot of the feature data can be found in appendix A.2. In this regard, the MAG model is very exciting for visually analyzing data.

Applying methods such as Node2Vec further allows for the creation of 2-dimensional node embeddings which can be used to create scatter plots. The scatter plots shown figure 4.1 show the node embeddings of the graph shown in figure 3.5. The scatter plot further helps to solidify the insights gained from the graph. A limitation of the graph for visual interpretation is, that one technically would need to consider the node connections. The large size of the graph however makes it impossible to visually follow the node connections for interpretation. Node2Vec considers the node connections/edges of the network and removes the mentioned limitation for visually interpreting the graph. Figure 4.1 shows, that the visual insights gained from the graph are still valid after considering the node connections. The Node2Vec algorithm is a beautiful dimensionality reduction technique, which successfully reduced the complexity of the graph shown in figures 3.4 & 3.5. In this sense, the MAG model in combination with Graph Representation Learning such as Node2Vec can be used as powerful cluster analysis tools.

5.2 Appropriateness of Semi-Synthetic Graphs for Machine Learning

The discussion in the previous section shows, that semi-synthetic graphs themselves can provide valuable visual insights. In this section it is discussed to what extent semi-synthetic graphs are appropriate for graph machine learning. The perhaps

most instructive evidence is provided by the Bank Telemarketing dataset presented in section 3.3. Graph machine learning fails to perform well on the graph generated from this dataset. It is shown, that the attributes chosen for the MAG model are unsuccessful for generating network structures that provided useful additional information which then could be exploited using graph machine learning. The Bank Telemarketing dataset is notoriously difficult due to the unbalanced distribution of the label. The dataset consist of approximately 90% of observations which did not invest in the short-term deposit compared to only 10% of the observations which did invest. For graph machine learning and also the standard machine learning methods, it is loss optimizing to classify most if not all observations as non-investors. When generating the biased graph shown in figure 3.2, a classification accuracy of $> 95\%$ is achieved which is a remarkable improvement. As mentioned in section 3.3, the biased graph which includes the label as an attribute cannot be used in a practical setting and is a form of cheating. Nevertheless it shows, that if a network structure can be generated which captures the label well, the problem of unbalanced label data can be overcome. In particular it shows, that the attributes must be capable of generating network structures that are at least in part predictive of the label. This is an interesting topic and warrants further research, as overcoming the problem presented by unbalanced label data is a relevant issue for machine learning.

The US Airline Passenger dataset is shown to be well suited for graph machine learning. The reasons for its success are in line with the previous discussion regarding the Bank Telemarketing data set. The attributes used for the MAG model generated network structures which are in part predictive for the label satisfaction of the US Airline Passenger data set. This is shown when looking at the satisfaction plots in figures 3.5 & 4.1.

The graph structure alone yields poor results when looking at the results shown in section 4.1 using Node2Vec. Taking a step back with a more generous perspective, it is nevertheless impressive that an accuracy of approximately 76% could be achieved for the training- and validation dataset which only considers 2-dimensional node embeddings. This however did not translate to the test graph for which the best accuracy was at only approximately 58%. In this sense, the model is only marginally more successful at the classification task than randomly guessing the classification.

Graph convolutional networks also achieve a classification accuracy of approximately 76%. This result is even worse when comparing it to Node2Vec. The GCN considers the network structure as well as the node features and should therefore be capable of outperforming Node2Vec. The GCN model clearly fails here and has the serious

limitation that it cannot be applied in an inductive setting to unseen graphs. GCN is a breakthrough model as it is one of the first successful Graph Neural Networks. Nevertheless, it performed poorly for the given datasets and is surpassed in terms of performance by more modern GNN methods such as GraphSage.

The Final graph machine learning method tested is GraphSage. This method yields very good results as shown in section 4.2.2. GraphSage is capable of training models which have a test accuracy of $> 94\%$. The aggregation strategies used for the GraphSage model further are successful in the order max-pooling $\geqslant$ sum-pooling $>$ LSTM $>$ mean. This ordering is based on the results presented in section 4.2.2 and is confirmed by the simulation results shown in section 4.2.3 and appendix A.1. When comparing the two main aggregation strategies of interest, max-pooling and sum-pooling, both methods almost yield identical test results. For the results shown in section 4.2.2 max-pooling is only more successful in classifying 1 more passenger as satisfied compared to sum-pooling. The test results are otherwise identical as shown in tables 4.7 & 4.6. The simulation results with 100 experiments for both max-pooling and sum-pooling confirm, that max-pooling yields marginally better results for the test graph with an average accuracy of 93.80% compared to 93.76% for sum-pooling. Sum-pooling is however more consistent, whereas max-pooling has a larger variance regarding the test accuracy. In addition, max-pooling suffered from two large negative test accuracy outliers during the simulation. Sum-pooling also suffered from two negative test accuracy outliers, which were however less pronounced. The GraphSage results show, that semi-synthetic graphs can indeed be appropriate for graph machine learning.

5.3 Graph Machine Learning Versus Standard Machine Learning

The overview table 4.9 shows that the results using GraphSage are competitive in comparison to the standard machine learning models. The results further confirm that the results using Node2Vec and the GCN are not competitive. GraphSage is shown to be the second best method behind the Random Forest classifier. The ANN model on average tends to yield marginally inferior test accuracy results as shown in the simulation results shown in appendix A.3. In addition, the ANN is more prone to over-fitting as shown in figure 4.9. With regard to the other standard machine learning methods, GraphSage clearly outperformed these methods.

This shows, that semi-synthetic graphs can be used for graph machine learning in a

competitive setting. The accuracy improvements one would hope for considering the rather tedious and difficult graph generation procedure is however not achieved. As shown in table 4.9, a simpler method such as the random forest classifier consistently outperforms the GraphSage model. In addition, a simpler standard ANN yields very similar results as GraphSage. In machine learning the principle of Occam's razor is often referred to for choosing appropriate models. Following this principle, simpler models are preferred if they yield similar/identical results compared to more complex methods. GraphSage using semi-synthetic graphs is most definitely the most complex method shown in table 4.9. For that reason, certainly the Random Forest Classifier is preferred as it yields better accuracies and is a more simple model compared to GraphSage. In addition, one could argue that the ANN, SVM and AdaBoost are preferential to the GraphSage model as well. These models are simpler with only marginally inferior results. For the evaluation one needs to also consider the context of the classification task. In a medical context, accuracies are of utmost importance where one is willing to use more complex methods to receive marginal accuracy improvements. For the purpose of gaining customer insights, the consequences for having marginally higher- or lower accuracies are less severe. In such a setting, the principle of Occam's razor can be more fully embraced which is why the ANN, SVM and AdaBoost might be preferred as they are simpler and more practical for application in a real-world setting. This is a subjective assessment and depending on the context might differ. Nevertheless, GraphSage yields the second best accuracies and remains a competitive model.

Chapter 6

Conclusion and Outlook

This chapter includes the conclusion and provides an outlook for future research.

6.1 Conclusion

The aim of this thesis is to assess to what extent semi-synthetic graphs based on real feature data are useful for machine learning. This aim corresponds to the research question stated in section 1.3. The results for answering this question are mixed and are based on the data presented in chapter 3, the results of chapter 4 and the discussion in chapter 5.

Graph machine learning on the graph created from the Bank Telemarketing dataset fails to overcome the problem of unbalanced labels. It however yields similar results in terms of accuracy as the standard machine learning models. In addition it is shown, that if a network structure can be created which corresponds to the label, that the problem of unbalanced label data could be overcome.

GraphSage performs well for the US Airline Passenger dataset and is second only to the random forest classifier in terms of accuracy. As discussed, following the principle of Occam's razor, the good results for the GraphSage model must be taken with a grain of salt. For the purpose of gaining customer insights, simpler models such as ANN, SVM or AdaBoost provide only marginally inferior results and are more practical and thus preferable for this setting. Given similar results for more sensitive applications such as medicine, the GraphSage model could be preferred as even marginal performance improvements can be of utmost importance. In short, the GraphSage yields competitive results in terms of accuracy, the results are however not superior enough to necessarily warrant the complex graph generation process and model complexity of GraphSage. Lastly, the graph convolutional network and Node2Vec are not successful for the given classification task. For that reason more

recent graph machine learning models should be tested and/or new models should be developed to expand the possibilities of graph machine learning.

The perhaps most interesting result is provided by the US Airline Passenger graph plots shown in figure 3.5. The MAG model successfully generates neighborhoods within the graph, where the nodes are grouped according to their similarity which respects the similarities between all attributes. This observation is further confirmed by the scatter plots shown in figure 4.1. The graphs and the scatter plots make it possible to visually interpret the relationships within the data. Standard scatter plots using the feature data do not allow for the generation of such insightful graphs/plots.

To summarize and provide an answer to the research question, it is shown that yes, semi-synthetic graphs can be useful for machine learning in a classification setting. The graph machine learning models are shown to be competitive and could potentially even provide a solution for overcoming the difficulties associated with unbalanced label data. This however requires the availability of a graph with a structure which corresponds to the label. The usefulness of semi-synthetic graphs are limited by the complexity associated with generating graphs and the general complexity of graph machine learning models. The graph based models further fail to outperform the standard machine learning methods and are ranked anywhere between the second to fifth best method depending on how one weights the trade off between model complexity and accuracy. Based on this, the hypotheses presented in section 1.3 can be answered as follows:

H1: Graph machine learning using semi-synthetic graphs fail to outperform standard machine learning methods. This hypothesis is thus rejected.

H2: Graph machine learning is shown to be a competitive strategy with competitive results in line with the results shown for the standard machine learning methods. This hypothesis is not rejected.

6.2 Outlook

The discussion in chapter 5 and the conclusion in section 6.1 provides interesting topics for future research. These topics include graph generation, cluster analysis on graphs and graph machine learning models.

Graph Generation

Creating semi-synthetic graphs using the MAG method is shown to be a viable

method. The graph is however sensitive to attribute selection and setting appropriate link-affinity probabilities. A fist step for resolving this issue was introduced by Kim & Leskovec (2011) as a follow-up to their MAG model. In this paper they propose a reverse model, in which given a real graph, the attributes and the link-affinity probabilities are estimated such that the attributes and link-affinity probabilities generate the observed real graph. This is however only a partial solution to the task given for this master's thesis. The estimated attributes do not correspond to real attributes/features. In this model, the attributes are estimated to fit the graph. An interesting topic for future research would be to develop a model that generates a semi-synthetic graph based on attributes which at the same time optimizes link-affinity probabilities such that the resulting graph adheres to network properties observed in real graphs. Perhaps these more realistic graph properties can improve the performance for graph machine learning. This is an area worth consideration as especially the degree distribution and centrality measures shown in figure 3.6 force graph machine learning models to consider a large number of neighbors. In addition, even when sampling, the neighborhood is bound to always have the same size, given the range of the degree distribution shown in figure 3.6. This is a potential limiting factor for graph machine learning, as it potentially makes it more difficult to distinguish- and make use of different structures withing the graph.

Cluster Analysis on Graphs

The graph plots shown in figure 3.5 reveal, that semi-synthetic graphs can be excellent tools for visualizing data. The amount of useful information provided for interpreting the data is a welcome and unexpected result. For understanding the relationships between the attribute data and the label, the graph plots yielded the most useful information for identifying relationships within the data. An interesting topic for future research could be to assess how common clustering methods such as k-means, fuzzy clustering or CLIQUE could be used for analyzing graphs. Perhaps better and more graph centric methods could be developed. An excellent overview of existing graph clustering methods is provided by Zhou et al. (2020). Their article can be used as an initial reference point for identifying new applications of existing graph clustering methods or for developing new models.

Graph Machine Learning Models

Graph convolutional networks and GraphSage are selected for this thesis as they are probably the two most well-known GNNs. There are however newer and more sophisticated models such as graph attention network (GAT) (Veličković et al. 2018) or graph isomorphism network (GIN) (Xu et al. 2019).

GAT models have the ability of identifying more- or less important neighbors in the graph. This is a useful ability and has been shown to improve performance in some cases. This ability is unfortunately most likely limited by the very large number of neighbors present in the US Airline Passenger graph as shown in figure 3.6. For that reason, more realistic graphs with smaller number of degrees should be generated as suggested in the previous graph generation paragraph.

GIN models are very well suited for distinguishing structures within the graph. The authors Xu et al. (2019) present the general GIN framework which can accept any type of features as inputs whilst ensuring that the aggregation function is injective. Given the injective aggregation strategy, it is shown that the GIN can be as expressive at distinguishing graph structures as the Weisfeiler-Lehman graph isomorphism test (Weisfeiler & Lehman 1968). Again, for such a model to work best, the number of degrees would most likely need to be smaller than currently present in the graph of the US Airline Passenger dataset. Given the large number of degrees, the neighborhoods of every node are currently of the same size when using the sampling strategy outlined for the GraphSage model.

Given a more realistic graph with a smaller number of degrees, it would be interesting to develop and assess, whether a new method which includes the attention mechanism of the GAT network with the isomorphic capabilities of the GIN method could be of use. This new model could for instance be tested on well understood benchmark graphs such as Cora (McCallum et al. 2000) or Citeseer (Giles et al. 1998).

Declaration

"I hereby declare - that I have written this master's 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 Faculty of Business and Economics of the University of Basel as stated in the faculty guidelines dated February 22, 2011."

Michael von Siebenthal, Martikel-Nr.: 2015-256-837, Date: July 12, 2021

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Giles, C. L., Bollacker, K. D. & Lawrence, S. (1998), Citeseer: An automatic citation indexing system, *in* 'Proceedings of the third ACM conference on Digital libraries', pp. 89–98.
URL: *https://dl.acm.org/doi/pdf/10.1145/276675.276685*

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*

Gundersen, G. (2020), 'The log-sum-exp trick', `https://gregorygundersen.com/blog/2020/02/09/log-sum-exp/`.

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*

KDD (2020), 'Stochastic training of gnn for node classification on large graphs'.
URL: *https://github.com/dglai/KDD20-Hands-on-Tutorial*

Kim, M. & Leskovec, J. (2011), 'Modeling social networks with node attributes using the multiplicative attribute graph model', *arXiv preprint arXiv:1106.5053* .
URL: *https://arxiv.org/abs/1106.5053*

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*

McCallum, A. K., Nigam, K., Rennie, J. & Seymore, K. (2000), 'Automating the construction of internet portals with machine learning', *Information Retrieval* **3**(2), 127–163.
URL: *https://doi.org/10.1023/A:1009953814988*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. & Bengio, Y. (2018), Graph attention networks, *in* 'Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - Mai 3, 2018'.
URL: *https://arxiv.org/abs/1710.10903*

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*

Weisfeiler, B. & Lehman, A. A. (1968), 'A reduction of a graph to a canonical form and an algebra arising during this reduction', *Nauchno-Technicheskaya Informatsia* **2**(9), 12–16.

Xu, K., Hu, W., Leskovec, J. & Jegelka, S. (2019), 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/hash/c5c3d4fe6b2cc463c7d7ecba17cc9de7-Abstract.html*

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*

Appendix A

Auxhiliary Results

A.1 GraphSage Simulation Results

The results for the GraphSage simulations are presented here. First, the results for sum-pooling are presented in figure A.1 and table A.1.

Metric	**Training Set**	**Validation Set**	**Test Set**
Average Accuracy	94.63%	94.78%	93.76%
	(0.46%)	(0.60%)	(0.37%)
Average Cross-Entropy Loss	0.1346	0.1346	0.1584
	(0.0109)	(0.0144)	(0.0096)

Table A.1: Simulation Results Sum-Pooling

The results for mean aggregation are presented in figure A.2 and table A.2.

Metric	**Training Set**	**Validation Set**	**Test Set**
Average Accuracy	94.75%	95.00%	93.63%
	(0.59%)	(0.56%)	(0.37%)
Average Cross-Entropy Loss	0.1353	0.1341	0.1665
	(0.0122)	(0.0171)	(0.0099)

Table A.2: Simulation Results Mean Aggregation

The results for LSTM aggregation are lastly presented in figure A.3 and table A.3.

Metric	**Training Set**	**Validation Set**	**Test Set**
Average Accuracy	95.64%	94.62%	93.75%
	(0.20%)	(0.69%)	(0.32%)
Average Cross-Entropy Loss	0.1130	0.1407	0.1630
	(0.0047)	(0.0120)	(0.0084)

Table A.3: Simulation Results LSTM Aggregation

Figure A.1: Simulations Results Sum-Pooling

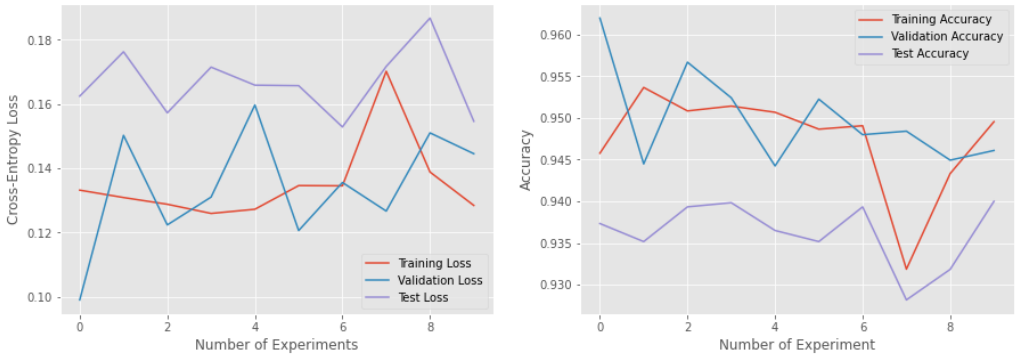

Figure A.2: Simulation Results Mean Aggregation

Figure A.3: Simulation Results LSTM Aggregation

A.2 Pairplot Feature Data

The pairplot of the feature data of the US Airline Passenger data is shown in figure A.4.

A.3 ANN Simulation

The results for the ANN were simulated using 10 experiments which trained the model for 400 epochs. The results are shown in figure A.5 and table A.4.

Metric	**Training Set**	**Validation Set**	**Test Set**
Average Accuracy	95.28%	94.74%	92.93%
	(0.61%)	(1.73%)	(1.07%)
Average Cross-Entropy Loss	0.1760	0.1433	0.2087
	(0.0268)	(0.0647)	(0.0569)

Table A.4: ANN Simulation Results

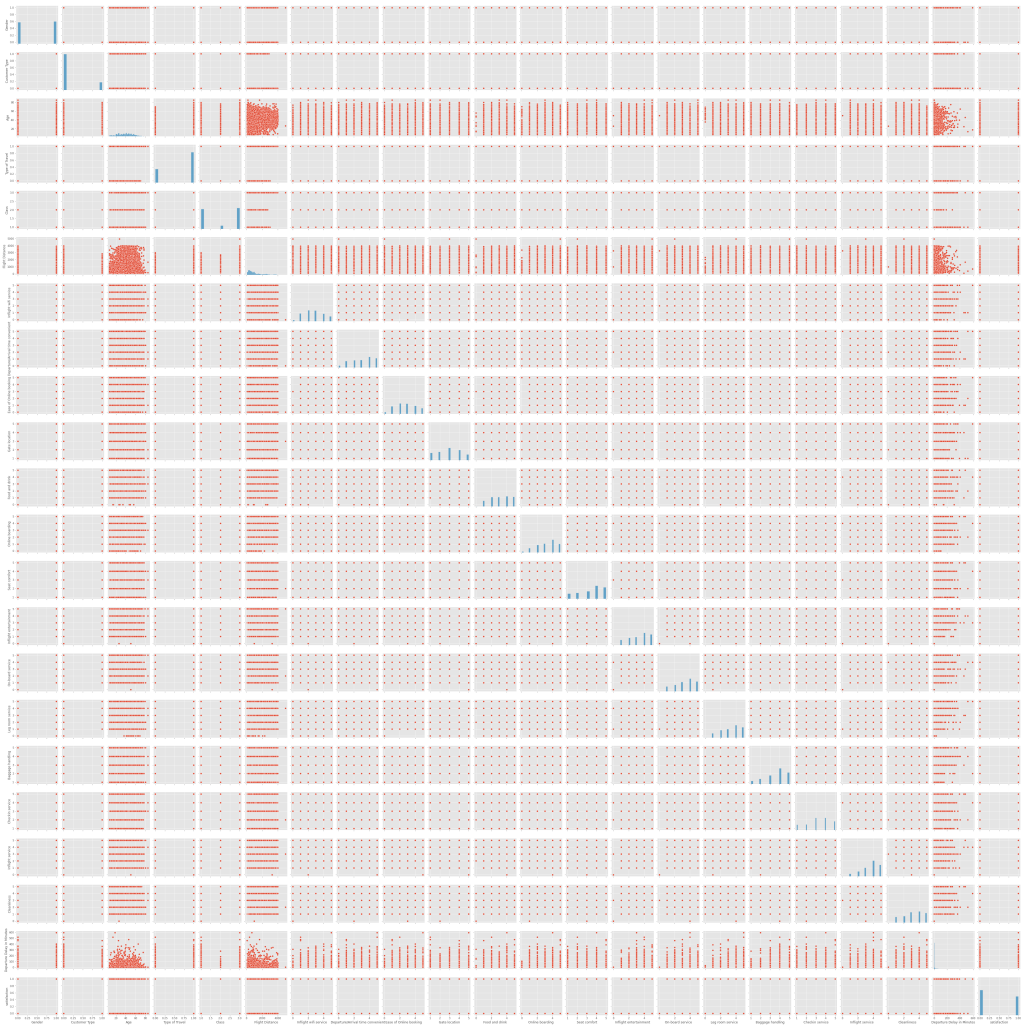

Figure A.4: Pairplot Feature Data US Airline Passenger Data Set

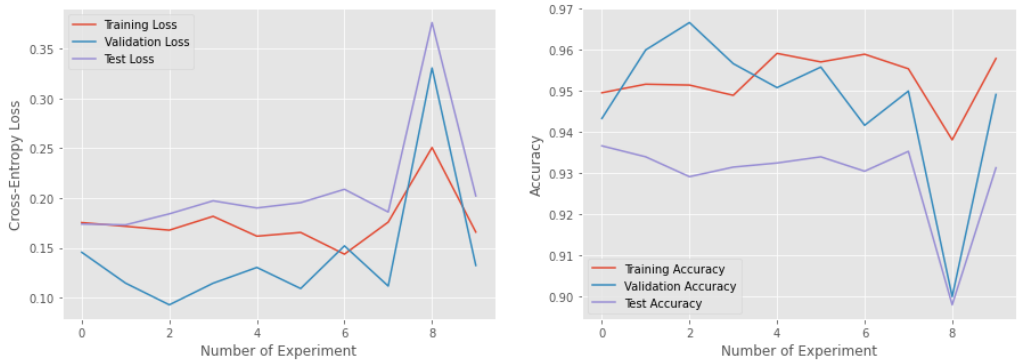

Figure A.5: ANN Simulation Results