FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Ondřej Měkota

Link Prediction in Inferred Social Networks

Department of Software Engineering

Supervisor of the master thesis:	doc. RNDr. Irena Holubová, Ph.D.
Study programme:	Computer Science
Study branch:	Computational Linguistics

Prague 2021

I would like to thank my advisor doc. RNDr. Irena Holubová, Ph.D. for her time spent reading my thesis and providing me with valuable suggestions and comments; and for her patience answering my questions. I would also like to thank my family for their support.

Title: Link Prediction in Inferred Social Networks

Author: Ondřej Měkota

Department: Department of Software Engineering

Supervisor: doc. RNDr. Irena Holubová, Ph.D., Department of Software Engineering

Abstract: Social networks can be helpful for the analysis of behaviour of people. An existing social network is rarely available, and its nodes and edges have to be inferred from not necessarily graph data. Link prediction can be used to either correct inaccuracies or to forecast links about to appear in the future. In this work, we study the prediction of missing links in a social network inferred from real-world bank data. We review and compare both verified and modern approaches to link prediction. Following the advancements of deep learning in recent years, we primarily focus on graph neural networks, and their ability to scale to large networks. We propose an adjustment to an existing graph neural network method and show that its performance is either comparable with or outperforming the original method. The comparison is performed on two social networks inferred from the same data. We show that it is relatively hard to outperform the verified link prediction methods with graph neural networks.

Keywords: inferred social network, link prediction, graph data, graph neural network

Contents

Introduction

Social networks represent human relationships. They can constitute relations between family members, co-workers, or people who often shop at the same stores. Many industries can benefit from the analysis of social networks. Marketing departments that would like to target advertisement to specific groups of people could analyse their past behaviour. A bank might want to personalise its products to the clients based on their observed interactions. Or health departments could improve tracing individuals infected with some disease.

However, it is quite presumptuous to assume that such social networks exist or that they are easily available. Due to various constraints in the real world, such as personal information protection laws, there is either not enough data to create a social network or there is enough data, but the data is protected and not easily accessible. Both nodes and edges have to be inferred from both relational and non-relational data (Brugere et al. 2016). The nodes can be either some entities in the source data or they might represent several clustered entities. Edges are inferred based on entity information. Entities which are similar in some way, for example, their attributes have similar values, may be connected by edges. There can be also temporal similarity – nodes having similar attributes in some timespan.

In this work, we analyse Inferred Social Networks (ISNs) and study link prediction methods in such networks. We have an access to real-world data from banking domain which serve as the main dataset used in our work. Link prediction is an essential problem in inferred social networks since the underlying data is often imperfect – some links are missing because of various reasons, e.g., network changes in time, and it would be worthwhile to predict which links are missing or which of them will appear or disappear in the future. Generally, algorithms used on social networks are well described (Chakrabarti et al. 2006); however, there is very little work on inferred social networks.

The objective of this thesis is to describe verified link prediction methods, adapt them to the real-world banking domain inferred social network, and perform experiments with them.

In particular, being given an ISN, we predict in which stores a client shops, who are his friends and household members and in which types of shops he shops most often. Based on an ISN inferred from data at the end of each month, we predict which points of sale (POSs) will be frequented by a given client in the next month.

With the advent of deep learning in the last several years, neural networks are more and more applied to graphs. In 2020, there have been 49 papers studying machine learning methods for graph data accepted to the International Conference on Learning Representations (Ivanov 2020). So called graph neural network (GNN) network architectures significantly outperform traditional link prediction methods on various graph datasets. However, there is not much work on utilising graph neural networks on complex social networks. In this work we adapt graph neural networks to inferred social networks and investigate whether they are able to achieve results comparable to well-known statistical methods.

The contribution of this thesis is the analysis and comparison of common link prediction methods and GNNs on inferred social networks; We propose an adjustment to the GNN architecture and show that it achieves results comparable to other GNNs in the link prediction task on an inferred social network. We also propose an alternative scoring function for link prediction using GNNs which outperforms existing methods on some tasks.

Structure of the Thesis

The thesis is split into the following chapters. Chapter 1 defines inferred social networks and explains the problems we try to solve, optimal results which could be achieved, and the constraints arising from the use of real-world data.

In Chapter 2 we define common GNNs methods used to train them.

Chapter 3 reviews traditional and deep learning based methods for link prediction.

Chapter 4 describes data with which we work in detail, it also explains how the inferred networks were created.

In Chapter 5 we define our GNN architecture and describe the GNN models which we adapted to our dataset and with which we experimented. We also describe used scoring functions.

Chapter 6 thoroughly describes our experimental settings, evaluation metrics, the selection of data and goes through the results.

1. Problem Definition

In this chapter we first define basic graph theory concepts and then describe common banking domain issues and propose how they could be solved using inferred social networks.

1.1 Basic Definitions

Before we define a social network, we first introduce basic graph theory notions.

Graph consists of a set of nodes (also called vertices or entities) connected by edges (also called relations or links). Graph can be *directed*, meaning its edges have a direction. In a *multigraph*, we allow more than one edge between two vertices and loops: edges going from a node back into it.

A *social network* is a graph, where nodes have types (such as "person", "shop", "account" etc.) and attributes or sometimes called features (such as "name", "age", "address", etc.). The edges also have types (distinct from the node types) and attributes, and they can be both directed or undirected. Nodes or edges of the same type have the same set of attributes. Attributes can be of generally any type: string, integer, decimal number, etc., or they can be empty (missing).

For example, a bank. Node types could be client, account, city, and ATM. The edge types would represent various interactions between the nodes, such as withdrawal of money from an account or client living in a city. Withdrawal could have the amount and time as attributes.

Let us define an *inferred social network*. Inferred social network is a social network created from relational and non-relational data (Brugere et al. 2016). Both nodes and edges are inferred using real world entity information. The nodes are either real world entities (e.g., people, places) or they might represent aggregated entities (e.g., shops selling the same type of products).

Edges are placed between nodes based on some predefined rules, consisting of the entity features, existing relations in the data, or a combination of both. Manually we can connect nodes that have the same features (e.g., people living at the same address). Statistical link prediction system estimates links based on some patterns in nodes (e.g., people often shopping in the same type of store).

Such a network can be inferred for a series of timespans, e.g., a network for every month.

1.2 Bank Problems

In this section, we describe the main banking problems[1]:

- *Loan* – predict whether a client will take a loan in the future
- *Default* – predict whether a client will miss a payment deadline

[1]Some information about bank problem comes from non-public documentation of research project TAČR TH03010276.

- *Fraud* – predict whether a client will try to acquire a loan based on misleading or untruthful information
- *Churn* – predict whether a client will prematurely pay off a loan or transfer the loan to a different bank

If we were given a social network of the clients, we might be able to solve these problems.

Firstly, we distinguish loan from mortgage. A loan is when a bank lends a small amount of money to a client with a relatively high interest rate, but the money need not be secured by something the client owns. On the other hand, a mortgage is money lent to a client to pay for a real estate. The interest rate of a mortgage is often lower than of a loan, but it needs to be secured by the real estate: in case the client does not pay their mortgage, the real estate is signed over to the bank.

The loan problem could be solved by analysing behaviour of clients forgoing the moment when they take a loan. Clients who need a loan may not have a lot of money in general and they may have less money imminently before taking a loan than they usually do. There are patterns in social networks determining whether clients take loans or not: people with lower education may be more likely to need a loan, employees of some employers can generally be less likely to take loans (e.g., employees of rich companies, such as Google, may not need a loan). In this work, we do not analyse the behaviour of clients taking a mortgage.

Married couples may have a higher chance of getting a loan since their combined income is usually higher than the income of one person, and it is more likely they will be able to repay the debt even in case one of them loses a job. However, when people are not married but live in the same household, the bank may not know about this because people can have a different corresponding address. If a person withholds the common household information from the bank, they could be given a loan even though the other person from the same household already might have been granted a loan. Were there enough similar cases, we could be able to predict that people live in the same household and prevent this type of fraud.

Similarly, there are triggering events that can help us predict client defaults. If several colleagues of a client suddenly lose a job, this client might be prone to losing a job too, resulting in missing a payment.

When a client pays off their debt prematurely, their real world friends might be more likely to repay their debt too, e.g., because there can be a better loan offer from another bank, and this information is passed among friends. Clients who had taken a loan in the past and whose friends or colleagues have subsequently got a loan with a lower interest rate may be more likely to switch banks.

1.3 Real World Data

In order to solve the above problems, an artificial social network has been inferred (Holubova et al. 2019). Both the nodes and the edges are artificially created from the original data.

There are missing values for some entities, e.g., some clients do not have their education status filled in, whether they have children, etc. Also, as the dataset consists of client data collected over several years, not all information has been acquired at all times. This makes the inferred social network inherently incomplete. Not only node attributes are missing, but so are some of the links between the nodes.

Another issue with solving the main bank problems is generally missing annotated data. There are very few frauds to be able to observe the behaviour of fraudsters. There are no exact data before which loan payments should be paid off, making any default prediction system hardly evaluable. Money in a bank account of a client may not be the only money he has. Employer of a client is not exactly known, and friends of colleagues are definitely not known.

In order to solve the main problems, we need to solve some smaller problems, for which we have enough data. As previously described, predicting household members would helpful for solving fraud and churn. Predicting friends of clients can help us foresee any defaults. We believe that predicting favourite types of shops, merchant category codes (MCCs), could be helpful for predicting who will take a loan. It is also desirable, although not an objective of this thesis, to fill in missing attributes of nodes (clients and other entities).

2. Graph Neural Networks

In this chapter, we provide an overview of GNNs and describe the most commonly used variations in more detail. Firstly, we describe the message-passing framework, which is the basis for most GNN architectures. Then we delve into more complicated GNN models used for solving complexity and memory issues with large graphs. We also review recent efforts in pretraining GNNs.

2.1 Message-Passing Framework

Graph neural networks is a class of machine learning algorithms (Scarselli et al. 2009). Given a graph, GNN can learn a deep representation of nodes and edges; it can be used to predict relations, classify nodes or edges, etc. Liu et al. 2020; Wu et al. 2020; Dwivedi et al. 2020 survey an extensive number of various graph neural network types and try to classify them and compare them.

Let us define a GNN. Let V be a set of nodes, and let $(v_i, v_j) \in E$ be a set of edges. Node features ${h_i}^0 \in \mathbb{R}^d$ for node v_i are d-dimensional real vectors [1].

Gilmer et al. 2017 show that GNNs can be understood as a special case of the message-passing framework. It has two phases: the message-passing phase and the readout phase.

During the *message-passing phase*, information is passed between nodes along the edges. Node representations $h_i^{(l+1)}$, on $(l+1)$-th layer, are updated using messages $m_i^{(l+1)}$ computed from the previous layer representations of the neighbouring nodes:

$$m_i^{(l+1)} = aggregate^{(l+1)}(\{h_j^l \,|\, v_j \in N(v_i)\}) \tag{2.1}$$

where $N(v_i)$ is the neighbourhood of node v_i (more precisely, a set of nodes, from which there is a directed edge to v_i), and *aggregate* (possibly trainable) function can be any function such as sum.

Hidden representation is computed from the aggregated representations of neighbours and hidden representation of the current node from the previous layer:

$$h_i^{(l+1)} = combine^{(l+1)}(h_i^l, m_i^{(l+1)}) \tag{2.2}$$

where the *combine* function is often a layer-specific learnable function.

During the *readout phase*, a feature vector is computed for the whole graph:

$$\hat{y} = readout(\{h_v^L \,|v \in V\})$$

where L is the last layer. In case one does not need to represent the whole graph, or a subgraph, the readout phase can be omitted. These cases include node classification and link prediction. See Figure 2.1 for an example of a graph and Figure 2.2 for an application of GNN on node v_0.

Most approaches do not consider edge features on the input (Kipf et al. 2017; Schlichtkrull et al. 2017; Hamilton et al. 2018; Xu et al. 2018), and only learn edge representations by aggregating the features of incident nodes.

[1] The 0 in h_i^0 means that the features are on 0-th layer, as we reuse this notation for describing representation at other layers.

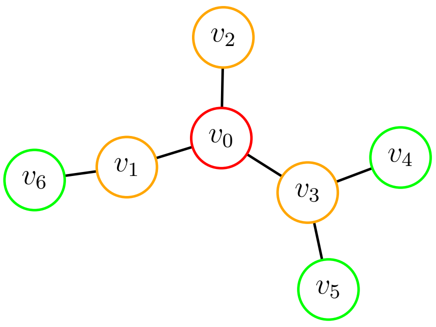

Figure 2.1: Example of a graph.

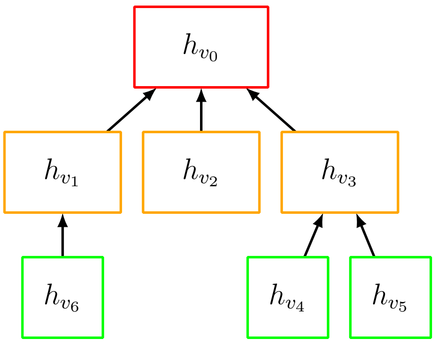

Figure 2.2: Forward propagation of GNN in graph from Figure 2.1 for node v_0. A similar computational graph is created for every node.

Schlichtkrull et al. 2017 propose a graph neural network suited for heterogeneous (or *relational*) graphs: relational graph convolutional network (RGCN). They compute node representation using Equation 2.3

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in N_r(v_i)} \frac{1}{c_{i,r}} W_r^l h_j^l + W_0^l h_i^l\right) \tag{2.3}$$

where $N_r(v_i)$ are the neighbours of node v_i connected using only edges of relation type r, $c_{i,r}$ is a normalisation constant and W_r^l are learnable parameters.

If the number of edge types is too big, they decompose the W_r^l matrices. Weight matrix decomposition helps the model generalise better when the number of edge types is large. It also prevents the model from overfitting. They propose two options to decompose the weight matrices.

Let $B \in \mathbb{N}$ be the number of bases. Basis decomposition is defined as:

$$W_r^l = \sum_{b=1}^{B} a_{rb}^l V_b^l \tag{2.4}$$

where a_{rb}^l, and V_b^l are both learnable parameters but only a_{rb}^l is edge type specific.

Block-diagonal decomposition is defined as a sum of submatrices on the diagonal of the weight matrix.

$$W_r^l = \bigoplus_{b=1}^{B} Q_{rb}^l \tag{2.5}$$

where $Q_{rb}^l \in \mathbb{R}^{(d^{(l+1)}/B)\times(d^l/B)}$ is edge and basis specific block-diagonal trainable matrix.

2.2 Large Graphs

Because GNNs can be very large, it can be challenging to train them due to many reasons. If the graph contains many nodes, edges, or attributes, it may not fit into the computer memory or the GPU memory.

If the graph is too dense (nodes have a high degree), there are issues with message aggregation because a lot of information has to be included in a fixed sized vector. This could make the model too generic as each node receives too much irrelevant information from its neighbourhood. Moreover, it causes a performance issue, as the gradient needs to be propagated to too many nodes, the time grows exponentially as the number of layers is increased. Such a problem prevents training deep models.

Hamilton et al. 2018 present a model called GraphSAGE. During the training of a node embedding, it randomly samples a small number of the adjacent nodes, making it feasible to use deep graph neural networks on large graphs. In addition to the sampling, they also use various embedding aggregation methods (the *aggregate* function from Equation 2.1): mean of the embeddings, max pooling, and an LSTM network (Hochreiter et al. 1997). As *combine* function, they concatenate the inputs and use a feedforward network on top of the result.

LSTM network seems like a rather counterintuitive choice of an aggregation function, as it is used for learning sequences (and it expects sequential data on the input). The neighbourhood of a node does not possess such ordering. The reason for using this network is that it does not need to have a predefined input size (such as a feedforward network). To overcome the ordering constraint whilst retaining the unlimited input size, they randomly order the nodes in the neighbourhood and empirically show that this heuristic is sufficient to make the model work.

Another possibility to address the computational and memory requirements for training GNNs is introduced by Chiang et al. 2019. Instead of training on the whole graph, they sample the nodes so that the graph induced on these nodes is as dense and possible, and they train the network only on this subgraph. They use graph partitioning software METIS (Karypis et al. 1999) to cluster similar nodes of the graph into a relatively large number of clusters. In each epoch, several different clusters are considered to reduce the bias of the clusters.

The results show that the training time and memory can be significantly reduced when a large number of layers is used. However, when the network has at most 2 layers, the time saving is insignificant because the graph clustering is also a complex procedure.

2.3 Other GNN Works

Xu et al. 2018 show that some graph neural network architectures such as (Kipf et al. 2017; Hamilton et al. 2018) have theoretically limited power when distinguishing different graphs. They compare the GNNs with a well-known heuristic for testing whether two graphs are isomorphic, the *Weisfeler-Lehman test* (WL test) (Leman et al. 1968), and show that GraphSAGE and GCN are not as expressive as the Weisfeler-Lehman test. On the other hand, all GNNs are at most as powerful as the WL test. Mainly, the *aggregate* and *readout* functions have to be injective to distinguish two topologically different neighbourhoods.

Given the shortcomings of these methods, they design a simple GNN architecture, graph isomorphism network (GIN), and theoretically prove that it is more expressive than GCN.

$$h_i^{(l+1)} = MLP^{(l+1)}((1 + \epsilon^{(l+1)}) \cdot h_i^l + \sum_{v_j \in N(v_i)} h_j^l) \tag{2.6}$$

where MLP means multilayer perceptron, by MLP we understand any feedforward neural network; $\epsilon^{(l+1)}$ is a layer-specific learnable parameter. They show that using ϵ fixed to 0 still achieves strong results, even outperforming a model where ϵ is learned.

Veličković et al. 2018 propose graph attention network (GAT), a model which uses the attention mechanism (Bahdanau et al. 2016) on messages passed from neighbours:

$$\begin{aligned} e_{i,j}^l &= LeakyReLU(a^T[Wh_i^l \parallel Wh_j^l]) \\ \alpha_{i,j}^l &= softmax_i(e_{i,j}^l) \\ h_i^{(l+1)} &= \sum_{v_j \in N(v_i)} \alpha_{i,j} W^l h_j^l \end{aligned} \tag{2.7}$$

where *LeakyReLU* (Maas et al. 2013) is non-linear activation function, a is a trainable parameter, and $[\cdot \parallel \cdot]$ is a concatenation operator. Actually, there is more than one attention computed for different parts of the input; these are called “attention heads”. Their results are concatenated. Attention mechanism provides an efficient way to “attend” to more important features and discarding the irrelevant ones; it is extensively used in natural language processing and computer vision. GAT shows performance comparable with GCN on node classification tasks.

2.4 Pretraining GNNs

Sometimes, we need to perform a task on node or edge types that exist in numbers too small for us to use them in a (semi)-supervised training. Instead, we would like to use some other node/edge types to pretrain the neural network while still being able to generalise on the original task. Following pretraining, we might either finetune the model on the small set of nodes or edges of the type in question or perform our task with only the pretrained model.

Z. Hu et al. 2020 propose a pretraining framework (GPT-GNN) inspired by the state-of-the-art language model, GPT (Radford et al. 2019). They pretrain the model by trying to reconstruct a graph corrupted by randomly removed edges and masked node attributes. GPT is an autoregressive model (a model that learns $P(x_{i+1}|x_1, ..., x_i)$) based on the transformer (Vaswani et al. 2017). They create a permutation of nodes and then try to decode the removed edges and node attributes in the permutation order from the existing part of the graph. The permutations are random so that the model does not learn any specific node ordering. GPT-GNN gains approximately 10% mean reciprocal rank in edge prediction task over a not pretrained model.

W. Hu et al. 2020 present a much simpler architecture than GPT-GNN. They perform a few pretraining tasks on node level: context prediction, masked node attribute prediction, and on graph level: structural similarity prediction and graph-level property prediction. The graph-level prediction tasks are not applicable to our case because we train our model on one large graph, whereas graph-level pretraining is suitable when training on a large set of small graphs. Context prediction classifies whether a neighbourhood of a node and context of a node belongs to the same node, where context is defined as an induced subgraph on nodes that are at least k_1 hops away from the node in question and at most k_2 hops away. Attribute prediction simply masks random attributes of a node and lets the model predict them. Generally, the model outperforms a not pretrained baseline by about $7 - 9\%$, however on some graphs, they show the pretraining underperforms the baseline.

3. Related Work

In this chapter, we describe both traditional statistical link prediction methods and more recent methods which use graph neural networks.

Social networks exhibit some characteristic properties. Links are more often formed incident to nodes with high degree (Bringmann et al. 2010). This enables one to use algorithms for link prediction which depend solely on this assumption.

3.1 Traditional Algorithms

The algorithms in these sections use a similarity measure between two nodes in a graph to predict a missing link. Despite their simplicity, they can achieve reasonably good performance (Liben-Nowell et al. 2003). One important disadvantage of these methods is their inability to exploit node attributes, a property of social networks which contains a lot of information about the structure. Another disadvantage of most of these methods is their strong assumption on forming a link: structural proximity of the edge's end nodes. P. Wang et al. 2015 survey state-of-the-art (at the time of writing) link prediction methods.

Bringmann et al. 2010 propose an association rule mining algorithm, Graph Evolution Rule Miner (GERM), for predicting the future evolution of a network (and hence link prediction). Association rule mining generally aims to find frequent rules or patterns in a database of sets of items, a detailed description of pattern mining algorithms is beyond the scope of this thesis; we refer the reader to a survey of pattern mining algorithms (Fournier-Viger et al. 2020).

Jaccard Coefficient is a measure used for estimating similarity between two sets (Jaccard 1912). In the context of social networks, it is used for link prediction. The score can be computed as follows:

$$Jaccard(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} \tag{3.1}$$

where $N(u)$ is a set of neighbouring nodes of node u.

Adamic et al. 2003 proposed a similarity statistic, **Adamic-Adar Index**, defined as follows:

$$AdamicAdar(u, v) = \sum_{w \in N(u) \cap N(v)} \frac{1}{\log |N(w)|}$$

Resource Allocation Index is similar to Adamic-Adar except it does not take the logarithm of the denominator:

$$ResourceAllocation(u, v) = \sum_{w \in N(u) \cap N(v)} \frac{1}{|N(w)|}$$

When the number of common neighbours of u and v is low, the difference between Adamic-Adar and Resource Allocation Index is insignificant (Zhou et al. 2009).

Preferential Attachment assumes that the probability of an edge is proportional to the number of neighbours of the nodes on its ends (Liben-Nowell et al. 2003):

$$PreferentialAttachment(u, v) = |N(u)|\,|N(v)|$$

Notice that preferential Attachment does not use common neighbours of n and v, which is the main difference from the previously described methods.

3.2 GNN-Based Link Prediction

There are several options how to perform link prediction using GNNs. The most straightforward is to use embeddings of source and target nodes and combine them to compute a link probability. Schlichtkrull et al. 2017 perform link prediction this way. The link prediction model is called DistMult (Yang et al. 2015). DistMult receives representations of two nodes (source and target), and edge type, and returns edge-type specific score of edge existence.

$$distmult(s, t, r) = \sum s \cdot w_r \cdot t \tag{3.2}$$

where s, t are the node representations of source and target nodes, respectively, and w_r is a learnable tensor for the edge type r.

Zhang et al. 2018 propose using the induced subgraph around the edge in question to estimate its likeliness. This method overcomes the limitation of most other methods, which is the assumption that two nodes are likely to link if they have common neighbours or similar features. They show that the network structure is approximately learned using an n-hop neighbourhood of a node. By sampling positive and negative edges and their n-hop neighbourhoods from the graph, they create their dataset.

Additionally, nodes are labelled according to their distance to source and target nodes of each edge. Source and target have label 1, nodes with distance 1 from both source and target have label 2, etc. Nodes that are more than n hops away from either source or target are given label "∞".

Let us define *line graph* $L(G)$ for a graph $G = (V, E)$. Each node in $L(G)$ represents an edge in G. Two nodes in $L(G)$ are adjacant if the corresponding edges are incident in G.

Following the methods described by Zhang et al. 2018, Cai et al. 2020 predict edges by creating a line graph from the original graph and converting the problem of link prediction to node classification (Cai et al. 2020). To preserve node attributes and node labels (which would otherwise be turned into edge attributes), they create edge features by concatenating the attributes of the neighbouring nodes in the original graph.

Rossi et al. 2020 propose a model for learning temporal events in graphs. An event is an attribute change or creation of a node or an edge at time t (deletion is not considered). Given a *continuous time* dynamic graph, the algorithm aims to learn the graph embedding at all times. Continuous time means that the graph is represented as a list of (large number of) events, as opposed to a discrete time graph which is a list of snapshots of the graph in time. The method is evaluated on link prediction on a network of Twitter interactions and it outperforms other link prediction methods on this highly dynamic graph.

4. Dataset

We are using data provided by a bank. The data spans over a period of 3 years and it contains anonymised information about clients, accounts, transactions, loans, and ATM withdrawals. The original data contains inconsistencies arising from the fact that the data were collected in various systems and by different banks (in the past, the bank was split into two banks). There are also general errors in the data, such as company accounts are sometimes assigned sex.

Two teams were working with the data, one from the Charles University (CU team) (Holubova et al. 2019) and the other from software company Profinit (PF team). Their approach slightly differs, the CU team infers a social network containing various types of nodes and edges. The PF team, however, creates several networks, each network containing only client nodes, with edges representing mutual similarities computed using various input data.

The CU team distinguishes a low-level and high-level view of the network. High-level view contains only clients as nodes with edges between them representing various kinds of relations. Low-level view includes also other entities, such as ATMs, shops, and employers. We work with both low and high-level view of the network. The nodes may also be clustered in order to reduce the size of the network, e.g., shops can be grouped based on their merchant category code (MCC). Each merchant (store or eshop) has been assigned an MCC; it identifies a "category" of products or services it sells, e.g., MCC 5462 are bakeries.

4.1 Dataset CU

This dataset is an inferred social network created by the CU team. The resulting social network is very large (see Table 4.1 for the number of nodes and Table 4.2 for the number of edges). Because of the lack of computational resources for estimating whether there should be an edge between two nodes (creating the high-level view), the dataset has been restricted. The remaining nodes in the smaller dataset are called "active". Column "Restricted count" in Table 4.1 shows the number of nodes of given type which were used for computing similarities. Nodes have various attributes of various types (strings, numbers, categories).

Client nodes were restricted to only those living in Prague. From other nodes which were restricted, only those with which there was an interaction more than k times were kept. The parameter k was set for each node type separately until an acceptable number of nodes was reached. For example, only eshops with at least 5 transactions were kept.

Each edge type has only one attribute, a floating point weight.

4.2 Dataset PF

Dataset PF is a dataset created by the PF team. It is a dynamic dataset containing client data for a period of two years. The network is computed for each month in the observed period. It contains only similarities of clients based on various information and their favourite POSs. POSs are places where a customer

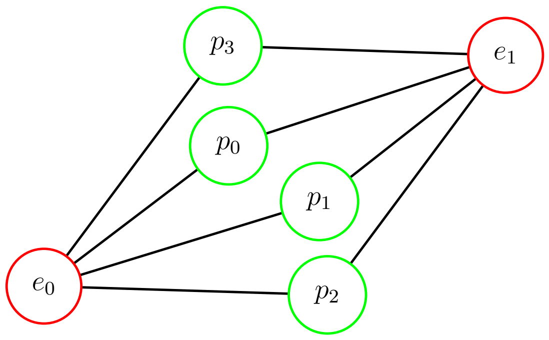

Figure 4.1: Edges between entities (e_0, e_1) and intermediates ($p_{0\ldots3}$)

pays for goods or services, the most obvious example is a shop.

Based on the data from each month, we could, for example, predict which POSs will be the client's favourite in the following months.

Every month $47-49$ thousand of client–MCC edges are added, and about the same number of them removed.

The client nodes are the ones described in Section 4.1.

4.2.1 Similarity Edges

We use three types of similarity edges. The first edge type, let us call it type A, is client similarity computed from the amount of money spent by the client in Czech stores in the given month, considering only stores where the client shopped at least three times. The second type of edges, type B, is client similarity computed from the number of ATM withdrawals in the last year, considering only ATMs where the client made at least 10 withdrawals. The third used edge type, type C, is computed using the sum of outgoing transactions in the last year.

The PF team computes the similarities as follows. They distinguish two types of nodes: entities and intermediates. Entities are the nodes between which the similarity is to be estimated, e.g., clients. Intermediates are selected types of nodes used to compute the similarity transitively. They compute the similarity between entities using transitive edges through intermediates. Entities sharing no intermediates have 0 similarity. Each entity has reflexive similarity 1. See Figure 4.1 for an illustration of entity–intermediate relations.

Each entity e has an *ability to connect* with intermediates (C_e), computed as an average of weights $v_{e,p}$ of edges incident to this node. Similarly, each intermediate p has also an ability to connect to entities, computed in the same way: C_p. For example, the average number of outgoing transactions weighted by the amount.

The similarity between an entity and an intermediate is evaluated as follows:

$$w_{e,p} = \frac{v_{e,p}}{\sqrt{C_e \cdot C_p}}$$

Similarity between two entities e_1, e_2 is w_{e_1,e_2}; it is evaluated as:

$$W_{e_1,e_2} = \sum_p \sqrt{w_{e_1,p} \cdot w_{e_2,p}}$$
$$w_{e_1,e_2} = \frac{W_{e_1,e_2}}{\sqrt{W_{e_1,e_2} \cdot W_{e_1,e_2}}}$$

Only edges with similarity at least 0.05 were kept.

Because the number of possible pairs is very large, the candidate edges are restricted using local-sensitive hashing (LSH) (Leskovec et al. 2020). LSH is a family of hashing functions that map similar points from a metric space to the same bucket with high probability. Dissimilar points are mapped to the same bucket with low probability. Using LSH makes the resulting network an approximation but it significantly speeds up the computation. More information on LSH can be found in book by Leskovec et al. 2020.

In Table 4.3, we provide the average numbers of edges that were computed each month.

4.2.2 POS Nodes

There are 60 951 unique POS nodes, with 43 unique MCCs, in 125 countries. Due to minor inconsistencies in the data, we were not able to exactly extract the cities in which these POSs are located. Nevertheless, after some semi-automatic post-processing, we arrive at are 11 883 unique cities and 51 363 unique POS names.

As the cardinality of the categorical features describing POSs is very large, we need to reduce it in order to make the training feasible and improve generalization. Because almost every POS has a different name, we decided to drop this feature altogether as it does not bring any valuable information for the majority of the POS nodes.

On average, there are 5.1 POSs in each city; however, in 60% of cities there is only one POS, in 80% of cities there are only 3 POSs; on the other hand, in each of the top 1% (= 119) of cities there is more than 58 POSs, and in the top 1 city there are 2 496 POSs. See Figure 4.2 for plot of the distribution of POSs in cities.

The distribution of POSs in countries is slightly more balanced. On average, there are 487 POSs in a country but the median is only 23. See Figure 4.3 for the distribution.

MCCs distribution is even more balanced. On average, each MCC is assigned to 1 417 POSs. Moreover, half of the MCCs are assigned to at least 340 POSs. See Figure 4.4 for MCC distribution.

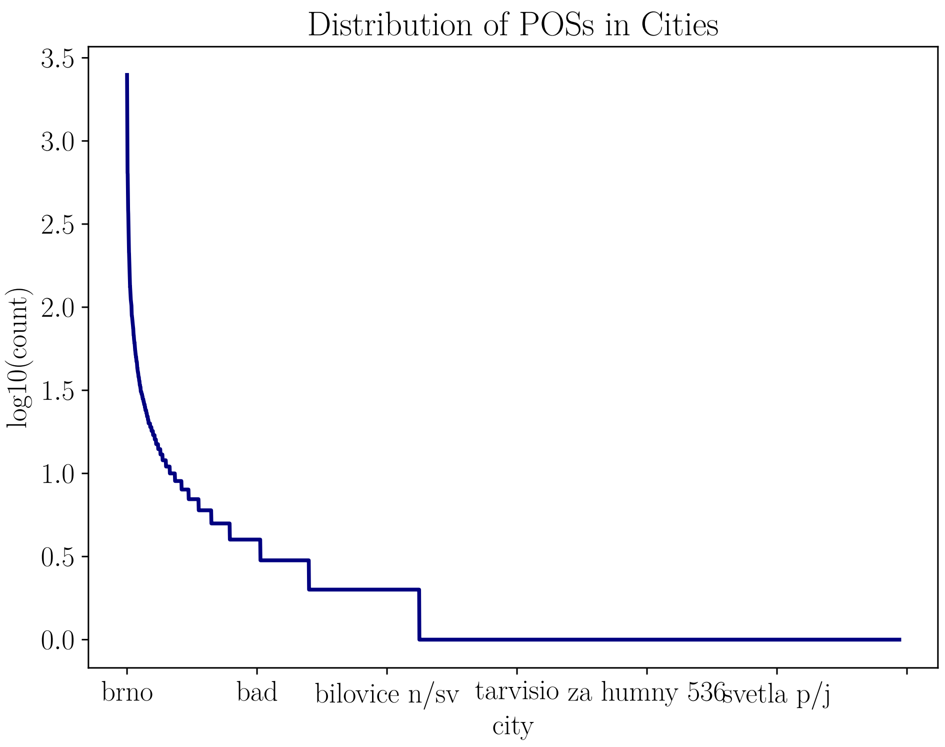

Figure 4.2: Distribution of POSs in cities.

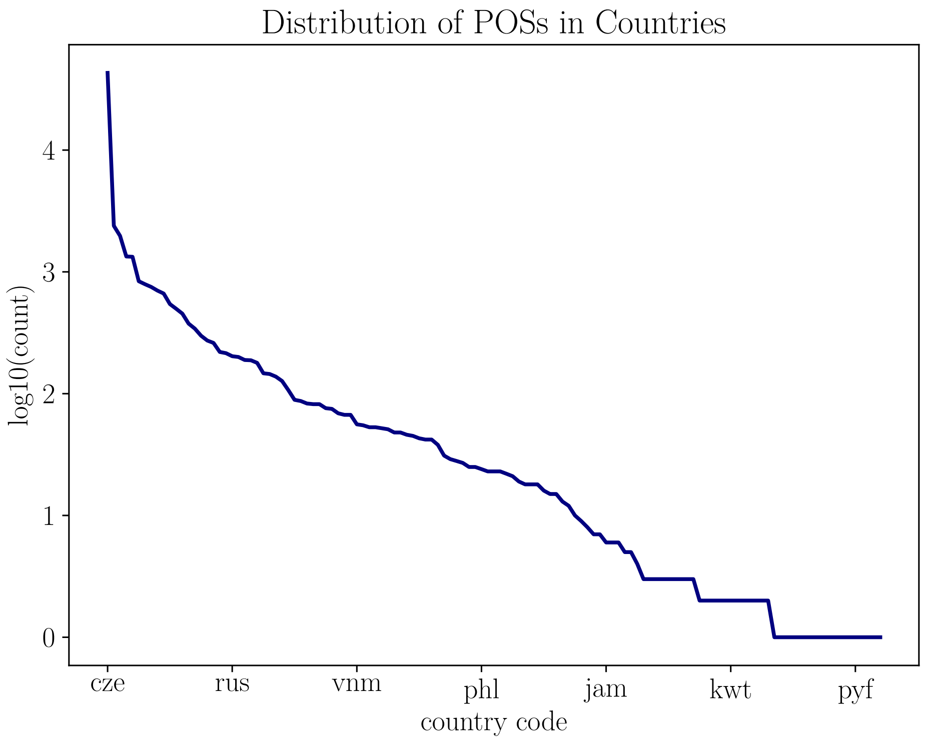

Figure 4.3: Distribution of POSs in countries.

Type	Variant	Count	Restricted count
Client		406 521	54 904
Counterparty	Internal	155 825	18 154
Counterparty	External	2 695 595	20 629
Employer		78 461	27 644
Shop	Store	840 237	33 308
Shop	Eshop	1 264 666	43 457
ATM		122 334	22 776
ATM v2		176 335	23 762
MCC	Code	1 320	602
MCC	Group	43	43
MCC	Class	17	17
Currency		260	260
Bank		57	57
Risk score		8	8
Risk class		6	6
Education		8	8
Social status		7	7
Marital status		5	5
Household members		10	10
Household children		9	9
Phone type		3	3
Segment		12	12
Address	Street	82 262	82 262
Address	Town part	15 094	15 094
Address	Town	6 258	6 258
Address	District	77	77
Address	Region	14	14
Zip code		2 670	2 670
Country		249	249

Table 4.1: Inferred social network nodes. Type column is the name of the node, Variant column represents a sub-type of the node, type and variant uniquely identify a node. Count column represents the number of nodes in the original data. Restricted count represent the number of nodes kept after preprocessing.

Type	Description	Weight	Count
Client - Client	Outgoing payments	sim	12 636 662
Client - Client	Outgoing domestic payments	sim	12 530 540
Client - Client	Outgoing foreign payments	sim	1 504 887
Client - Client	Incoming payments	sim	13 316 522
Client - Client	Incoming domestic payments	sim	13 415 303
Client - Client	Incoming foreign payments	sim	1 380 568
Client - Client	Salary transfer	sim	5 045 505
Client - Client	SIPO payments	sim	849 035
Client - Client	Permanent payments	sim	4 624 085
Client - Client	Loan receiving	sim	3 337 333
Client - Client	Loan instalements	sim	3 268 742
Client - Client	Debit card transactions	sim	11 147 509
Client - Client	Debit card payment	sim	10 374 993
Client - Client	Debit card payment with cashback	sim	123 149
Client - Client	ATM withdrawal	sim	10 563 408
Client - Client	ATM withdrawal domestic	sim	9 316 026
Client - Client	ATM withdrawal foreign	sim	3 142 220
Client - Client	MCC class	sim	10 571 834
Counterparty - Counterparty	Incoming counterparty payments	sim	5 955 885
Counterparty - Counterparty	Outgoing counterparty payments	sim	5 372 484
ATM - ATM	ATM witdhdrawal in Czechia	sim	526 603
Country - Country	Payments and withdrawals	sim	20 503
Client - Client	Household members	1	29 976
Client - Client	Friends	1	37 743
Client - Counterparty	Sending money to counterparty	count	4 100 416
Client - Counterparty	Receiving money from counterparty	count	5 604 070
Client - Bank	Communicating with bank	count	2 240 956
Client - Country	Spending money in country	count	427 189
Client - Shop	Shopping in an eshop	count	4 124 379
Client - Shop	Shopping in a store	count	19 837 284
Client - ATM	Withdrawing from an ATM	count	5 313 151
Client - Currency	Making transactions in currency	count	394 831
Client - Employer	Receiving money from an employer	count	304 333
Client - MCC	Shopping in shops with MCC class	count	2 464 846
Client - MCC	Shopping in shops with MCC code	count	7 290 679
Client - Risk Score	Risk score	1	108 387
Client - Risk Class	Risk class	1	137 612
Client - Education	Education	1	133 075
Client - Social Status	Social status	1	130 852
Client - Marital Status	Marital status	1	133 336
Client - Household Members	Household members	1	130 852
Client - Household Children	Household children	1	135 425
Client - Phone Type	Phone type	1	135 425
Counterparty - Segment	Counterparty segment	1	675
Client - Town part	Client town part	1	403 752
Client - Town	Client town part	1	384 848
Client - ZIP	Client ZIP	1	478 975
Client - Country	Client country	1	426 208
ATM - Town	ATM Town	1	4 205
ATM - Country	ATM Country	1	122 263

Table 4.2: Inferred social network edges. The type and description columns uniquely identify an edge. The weight column denotes how the edge weight was computed: either a similarity score (sim), number of edges connecting given nodes (count), or just identity edges (1). The weight of the edge is normalised so that the value is near 0.

Type	Type	Average month edges
Client - Client	A	7 633 475
Client - Client	B	1 665 577
Client - Client	C	4 842 046

Table 4.3: PF Dataset similarity edges

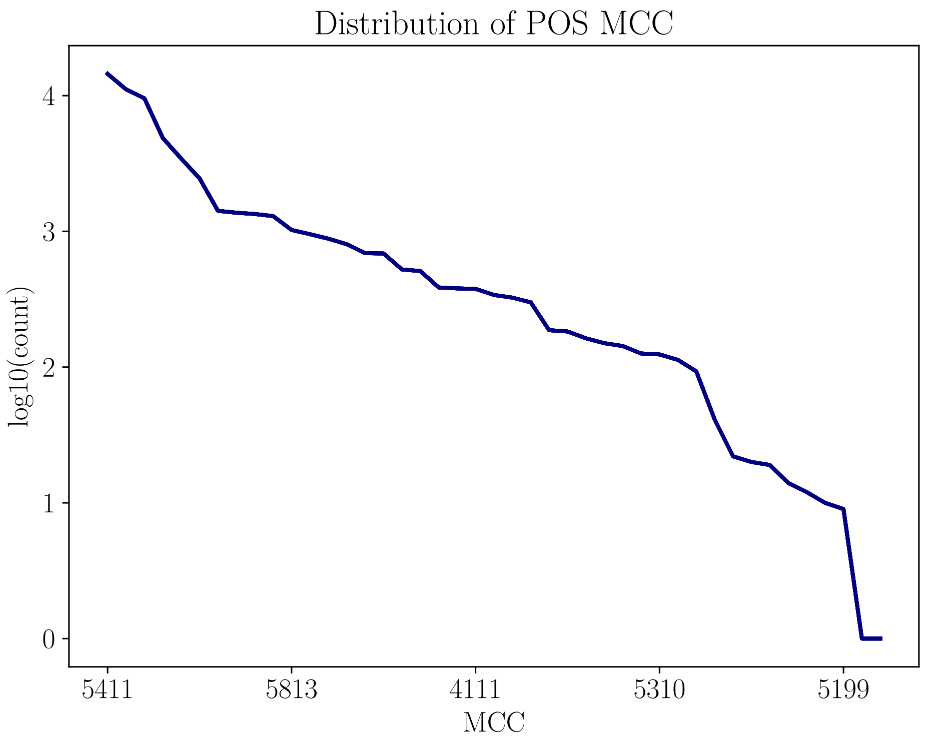

Figure 4.4: Distribution of POS MCC.

5. Model

In this chapter, we describe the settings and the variations of the architectures (which themselves are described in Chapter 2) with which we experiment. We experimented with several GNNs, mainly GCN, GIN, GAT, GraphSAGE, and our model, the DeepGCN, and with traditional link prediction methods: Jaccard Coefficient, Resource Allocation Index, Preferential Attachment, and Adamic Adar. We also describe the scorers used to compute the score of a candidate edge using GNNs.

5.1 Statistical Models

Among the traditional, statistical link prediction methods, we chose Jaccard Coefficient, Resource Allocation Index, Preferential Attachment and Adamic Adar. These methods are implemented in the NetworkX library (Hagberg et al. 2008) and are easy to use.

A disadvantage of these methods is that they are unable to process graphs with more than one type of nodes or edges. This disadvantage is especially limiting if we want to predict edges between nodes of the same type, such as the client–client household edges. A way to deal with this issue is to select and filter the input graph for predicting each type of edge or merge multiple edges.

5.2 GNN Models

We experiment mainly with GCN, GIN, GAT, GraphSAGE, and DeepGCN. Great advantage, contrary to the traditional link prediction methods, is the ability to process various types of nodes and edges. Neural networks are known to learn which features are important and which not, making them easier to use without expert domain knowledge. One does not need to spend as much time preprocessing the data as one would have to with the statistical methods.

We trained the model using mini-batches. At each step, a set of edges of a given type is sampled from the graph, and the surrounding subgraph is extracted for each hidden layer in the network. This method significantly increases training speed because of two reasons. Firstly, the gradient needs to be propagated through fewer nodes, and secondly, there are more backpropagation steps per epoch (this holds for any mini-batch training).

5.2.1 Scorers

We use two types of scoring functions. The first is DistMult, described in Equation 3.2 The likelihood of edge existence can be obtained by computing sigmoid of the result:

$$score(s,t,r) = \frac{1}{1+e^{-distmult(s,t,r)}} \tag{5.1}$$

The other method for computing edge score we tried is to concatenate the representation of the end nodes of an edge and run a neural network on it:

$$score(s,t,r) = MLP_{r,out}(LeakyReLU(MLP_{r,in}([s \parallel t]))) \tag{5.2}$$

where $[\cdot \parallel \cdot]$ is concatenation. We call this scorer the DeepScorer.

5.2.2 Training

We train the model by minimising cross entropy between the scores and labels:

$$H(p,q) = -\sum_{x \in \mathcal{X}} p(x)\ log\, q(x) \tag{5.3}$$

where $\mathcal{X}$ is the set of training examples and p, q represent real and predicted distributions.

We use negative sampling (Mikolov et al. 2013) during training. That is, for each existing (positive) edge, we sample n non-exiting (negative) edges, where n is a small number ($\sim 5-10$). The negative edges are sampled so that the source node is included in some edge from the positive edges, and the destination node is selected randomly (within the given node type). Compared to sampling both ends of the negative edge randomly, fixing one end makes the task more realistic and harder because the edge is more likely to exist. In order to prevent the model from overfitting on the existence (or non-existence) of positive (or negative) edges, we remove the positive edges from the graph in each mini batch.

Before training, we set aside a portion of the edges for evaluation, and completely delete them from the training graph, so that the neural network can not learn anything about their existence. On the other hand, the network might learn that these edges do not exist in the graph (through negative sampling); however, since this holds for all the GNNs we are comparing, and since the chance of selecting a high number of these edges as negatives is very small, it does not matter much.

For some tasks, we experiment with pretraining. During pretraining we train the link prediction not only on the type of the edge we want to predict but on more types of edges. Even though there are better approaches (Z. Hu et al. 2020; W. Hu et al. 2020), this one is relatively easy to implement and according to W. Hu et al. 2020 still might yield an improvement over the original task. Despite the fact that Z. Hu et al. 2020 report slightly better performance over non-pretrained models, we decided not to implement their architecture. Firstly because the improvement is quite small, and also because training transformer architecture very time consuming without a GPU (even a very scaled down version of it), and even with a GPU the training is extremely unstable (Popel et al. 2018).

5.2.3 DeepGCN

We propose an adjustment to the basic GCN architecture. Firstly, instead of applying a trainable matrix to the neighbour representations during aggregations, we use a more expressive multilayer neural network. Inspired by GIN, we also use the representation of the center node itself. The difference from GIN is that we also apply a neural network before adding it to the results. Another neural network is applied to the result:

$$h_i^{(l+1)} = MLP_{out}^{(l+1)}(MLP_{self}^{(l+1)}(h_i^l) + aggregate_{v_j \in N(v_i)}^{(l+1)}(MLP_{neigh}^{(l+1)}(h_j^l))) \tag{5.4}$$

where *aggregate* function is either sum, mean, or max. By using a deeper neural network, we believe that the model will be able to learn more complex transformations. We assume that using MLP_{self} could improve shallow GNNs because without it the model would sum the node features with neighbour representations (as it is in GIN) instead of the transformed features. MLP_{neigh} only provides bigger representational power than a trainable matrix. MLP_{out} further improves the node representation by applying a transformation on the sum.

We call this model the DeepGCN.

6. Experiments

In this chapter, we report the results of the experiments we have done. Generally, they differ in the input data or the model architecture. The experiments aim to solve problems described in Chapter 1.

This chapter is divided into several sections. In Section 6.1, we analyse the dataset size with respect to available computational resources. Section 6.2 describes the metrics used to compare the results. Section 6.3 provides an overview of our experimental setup and lists the specific parameters with which we experiment. In the following two Sections (4.1, 6.5), we describe the experiments that were performed on the two datasets, and we provide and discuss the results. In Section 6.3.2 lists the particular versions of software used in the implementation. Finally, in Section 6.6 we discuss the failed attempts at implementing some GNN models.

6.1 Notes on Data

To the best of our knowledge, no framework for working with graph neural networks (M. Wang et al. 2020; Grattarola et al. 2020; Fey et al. 2019) is able to process graphs larger than memory. Generally, to apply a graph neural network on a graph, the graph needs to be several times (~ 10, depending on its sparseness and the number of the training edges) smaller than the available RAM. The mentioned frameworks store the graph in special formats, e.g., various types of sparse matrices in order to speed up computation. The storage of the graph can be configured, but it still might be infeasible to use a GNN, because during backpropagation the gradient needs to be remembered for each node/edge for each layer in the network and then applied, in parallel, to the network weights.

In addition, because the dataset contains anonymised information about real world clients of a bank, it has to stay on a predetermined secured server. The server has 64GB of memory and a single CPU. Although the server resources could be increased to some extent, it still would not suffice for the task at hand. We experiment with various graph sizes and training set sizes.

In Table 6.1, we provide a comparison of RAM allocation and time measurements while using different sized datasets. As can be seen, when we use the dataset with 16 million edges, the RAM can rise up to 56GB (even if only 1-layer GNN is used). Note that in the original CU dataset, there are almost 200 million edges.

The number of edges is not the only variable influencing the usage of RAM. It is also the sparseness of the graph. The denser the graph, the more RAM is needed for training. Meaning, the fewer nodes per fixed number of edges are in the graph, the more edges are needed for propagating the gradient.

Also, this data comes from a relatively small bank with approximately 400 thousand clients. If we were to perform the same experiments on data from a larger bank, such as Česká Spořitelna, which has approximately 4.5 million clients (Česká Spořitelna 2020), the hardware resource would not suffice for the whole dataset.

Note that the time in Table 6.1 is influenced by the size of mini batch, we

Edges	Nodes	Train Edges	Data	Layers	RAM	Time
16 473 128	224 144	683	Static	1	8GB	7s
16 473 128	224 144	683	Static	2	19GB	45s
16 473 128	224 144	2 556 047	Static	1	56GB	675s
8 271 701	256 261	242 995	Dynamic[1]	2	35GB	55s

[1] This run is repeated for each month in the dataset.

Table 6.1: RAM consumption and time of training on various sizes of graph and training set. Edges column denotes the number of edges in the graph. Nodes column denotes the number of nodes in the graph. Train Edges column denotes the number of edges used for prediction. Data column denotes whether the dataset is static or dynamic. Layers column denotes the number of GNN layers. Time column denotes the number of seconds it takes to train a GCN model with DistMult predictor on the dataset for one epoch.

use size 64 for the first two experiments and 8 192 for the last two. Even though it makes it seemingly incomparable, we think it much more resembles the real situation, as for larger training sets, larger batch sizes are used. Also, because we train the model on a CPU, the training time does not linearly scale with batch size, as it would (to some extent) on a GPU.

To make our task feasible, we could either split the network into several pieces (graph partitioning) or omit some edges in the graph.

Graph partitioning assigns each vertex of a graph into one of c clusters. A partitioning that aims at the smallest number of edges between partitions is called *the minimum c-cut.* For a fixed c the problem is solvable in $O(n^{c^2})$, where n is the number of vertices (Goldschmidt et al. 1988). Nevertheless, graph partitioning is computationally demanding and the clustered graphs lose a lot of information, because a lot of inter-partition edges are deleted.

We chose to limit the number of edges between nodes by removing the most improbable ones – we are dealing with inferred social network, so improbable edges are therefore well defined.

6.2 Evaluation Metrics

Because we want to predict new edges in a network where it is much more likely that there is no edge between two random nodes, accuracy is not a suitable metric. A simple baseline could easily outperform a good model when measured by accuracy by simply classifying a candidate edge as non-edge. Instead, we use the average precision (AP) as our main evaluation metric.

Let us define precision and recall:

$$precision = \frac{tp}{tp + fp}$$
$$recall = \frac{tp}{tp + fn}$$

where tp, fp, fn is the number of true positives, false positives, and false negatives, respectively. AP then averages precision across all recall levels.

We also evaluate the models using the whole precision-recall curve and observing precision at various recall levels. Interesting recall levels are those where the precision begins to decrease rapidly. Precision at both high and low recall levels is interesting. At the low recall level, precision is usually high, meaning a small number of links is predicted very accurately, and a model with such predictive capability might be used without (or with a very low) human interaction. On the other hand, high recall levels have lower precision, but they still can be used for assisting humans in revision of the predicted links by greatly reducing the number of candidate links to review.

6.3 Experimentation Workflow

In this section, we describe the process which lets us acquire the final hyperparameters used in the models. We also describe the specific settings of individual models with which we experimented.

Even though we believe our experimental process is adequate, it is by no means exhaustive, and we cannot possibly guarantee our results are the best that can be achieved on the data, as there are hundreds hyperparameters to be set and due to the computational limitations, we have to search for them more manually than automatically (using a hyperparameter search algorithm (Yu et al. 2020)). Our hyperparameter selection process starts with a relatively small grid search for a subset of parameters — learning rate, batch size, number of hidden layers, the dimensionality of hidden layers, and hidden layer activations. This search is run for every GNN model we use. Based on the results, we disregard the worst performing hyperparameters and continue searching manually. This process is repeated on some smaller sets of parameters.

We further experiment with the learning rate decay. We started with lowering the learning rate by some factor every few epochs (one-step decay), which we further improved by a custom learning rate schedule by performing the decay at a selected epoch. We also used a technique that reduces the learning rate once a selected metric (loss, average precision, etc.) stops improving; in Pytorch (Paszke et al. 2019) it is called ReduceLROnPlateau. However, as the training progresses, metrics usually do not improve at every step, sometimes they slightly decrease between two epochs, and one does not want to decay the learning rate at every small decrease of the metric. Setting the tolerance for the metric variance for the ReduceLROnPlateau is difficult. After some experimentation with ReduceLROnPlateau, we decided not to use it, as it is possible to achieve the same results using a predefined learning rate decay schedule without the tiresome work of adjusting ReduceLROnPlateau.

6.3.1 Hyperarameters

In this section, we describe the hyperparameters with which we experiment. Mainly, we tuned the hyperparameters on the prediction of favourite MCCs.

The final hyperparameters are in Table 6.2. Generally, these parameters are used for all tasks, unless they are overwritten in the respective task-specific tables: 6.3, 6.4, 6.5, 6.6, and 6.7.

We have experimented with the following parameters:

Parameter	Value
multigraph aggregation[1]	sum
parameter initialization	Xavier[2]
batch size	8 192
use edge weights	False
trainable embeddings[3]	True
embedding dimension	29
hidden layer dimension	32
layer activations	none
negative sampling factor	10
pretraining non-evaluation edges[4]	100 000
GCN *aggregate* function	sum
DeepGCN *aggregate* function	sum
GraphSAGE *aggregate* function	max
GraphSAGE dropout	0.0
GIN *aggregate* function	sum
GIN learn epsilon	True
GIN initial epsilon	0
GAT number of heads	1

[1] Aggregation function of incoming edges with various types.

[2] Glorot et al. 2010

[3] Nodes that do not have any features use trainable embeddings.

[4] The edge types on which the evaluation is not be performed.

Table 6.2: Default hyperparameters. These parameters are used for all methods and tasks unless they are overridden in the tables following.

- The GNN model: {`GCN, GIN, GraphSAGE, GAT, DeepGCN`}
- The scorer: {`DistMult, DeepScorer`}
- The number of layers: $\{1, 2, 3\}$
- The activations of layers: {`none, ReLU, LeakyReLU, GELU, softplus`}
- The dimensionality of the layers: $\{16, 32, 64, 128\}$
- The initial learning rate: 10^{-5}–10^{-2}
- The learning rate decay method: {`ReduceLROnPlateau, one-step, schedule`}
- The learning rate schedule
- The learning rate decay factor: 0.1–0.5
- The batch size: 64–16384
- The weight of the loss for positive edges: $\{1, 1.2\}$
- The aggregator for different edge types: {`sum, mean, max`}

Model	Dataset	LR	LR schedule	aggregate
GAT	D-NoSim	$6 \cdot 10^{-4}$	`[60, 100]`	sum
GAT	D-All	$8 \cdot 10^{-4}$	`[40, 80, 110]`	sum
GraphSAGE	D-NoSim	$2 \cdot 10^{-4}$	`[20, 60]`	sum
GraphSAGE	D-All	$6 \cdot 10^{-3}$	`[20, 40, 60, 80]`	mean
GIN	D-{NoSim|All}	$4 \cdot 10^{-5}$	`[20, 60]`	sum
DeepGCN	D-{NoSim|All}	$2 \cdot 10^{-4}$	`[50, 100]`	sum
GCN	D-NoSim	$2 \cdot 10^{-4}$	`[50, 100]`	sum
GCN	D-All	$2 \cdot 10^{-4}$	`[50, 150]`	max

Table 6.3: Specific hyperparameters for the client–MCC edge prediction task.

Model	Dataset	Scorer	LR	LR schedule	batch size
GCN	D-{NoSim|All}	DeepScorer	$1 \cdot 10^{-4}$	`[45]`	64
GCN	D-{NoSim|All}	DistMult	$5 \cdot 10^{-5}$	`[40]`	64
DeepGCN	D-{NoSim|All}	DeepScorer	$5 \cdot 10^{-4}$	`[50, 100]`	64
DeepGCN	D-{NoSim|All}	DistMult	$5 \cdot 10^{-4}$	`[40]`	64

Table 6.4: Specific hyperparameters for the client–client household edge prediction task.

- The number of negative edges factor: $\{3, 5, 10, 20\}$
- Using edge weight: {`yes, no`}

On top of these parameters, we experimented with parameters specific to the GNN architecture.

GIN

As an *aggregate* function in the GIN model, we use a linear layer with LeakyReLU, with 0.01 negative slope, 0.2 dropout (Srivastava et al. 2014), and another linear layer without activation. We let the network learn ϵ from Equation 2.6.

We experimented with various dropout values: $\{0, 0.2, 0.5\}$, fixing ϵ or making it trainable. We also tried various networks for the *aggregate* functions, generally experimenting with a series of linear layers followed by an activation {`linear, ReLU, LeakyReLU`} and/or dropout.

GraphSAGE

We tried different aggregators of the embeddings: sum, mean, max, or LSTM layer.

GAT

We experimented with the number of attention heads: 1 or 2. We found that one attention head works best.

Model	Dataset	Scorer	LR	LR schedule
GCN	D-All	DistMult	$1 \cdot 10^{-3}$	`[40, 80, 100]`
GCN	D-NoSim	DistMult	$5 \cdot 10^{-4}$	`[15, 40]`
GCN	D-{NoSim|All}	DeepScorer	$5 \cdot 10^{-4}$	`[15, 40]`

Table 6.5: Specific hyperparameters for the client–store edge prediction task.

Model	Dataset	Scorer	LR	LR schedule	aggregate
GCN	D-NoSim	DeepScorer	$1 \cdot 10^{-4}$	`[45]`	sum
GCN	D-NoSim	DistMult	$1 \cdot 10^{-4}$	`[30]`	max
GCN	D-All	DeepScorer	$5 \cdot 10^{-4}$	`[50, 100]`	sum
GCN	D-All	DistMult	$1 \cdot 10^{-4}$	`[30]`	max
DeepGCN	D-NoSim	DeepScorer	$1 \cdot 10^{-4}$	`[30]`	sum
DeepGCN	D-NoSim	DistMult	$1 \cdot 10^{-4}$	`[30]`	sum
DeepGCN	D-All	DeepScorer	$1 \cdot 10^{-4}$	`[45]`	sum
DeepGCN	D-All	DistMult	$1 \cdot 10^{-4}$	`[45]`	sum

Table 6.6: Specific hyperparameters for the client–client friends edge prediction task. All models in this task have batch size 64.

DeepGCN

We have experimented with various neural networks for MLP_{self}, MLP_{neigh}, and MLP_{out}. Generally, it was always a fully connected feedforward neural network followed by an activation and/or dropout. It turned out, that the RAM consumption grows very quickly with any additional parameters in the GNN layers. In the end, we used a simple 32 dimensional linear layer without any activation nor dropout.

Pretraining

When we perform pretraining on any dataset, we use all the edge types but client similarity edges. This is to reduce the training time. We reduce the number of training edges. We sample at most 10^5 of edges of types on which the evaluation will not be performed. This makes the training faster.

During training, we take the average of losses computed for each edge type. We also experimented with adding more weight to the edge types with fewer edges; however, it did not show any improvement.

6.3.2 Implementation

The GNN models are implemented in the Deep Graph Learning (DGL), version 0.5 (M. Wang et al. 2020), a Python library for implementing GNNs. DGL is built on top of a well known deep learning library, PyTorch, version 1.7.1 (Paszke et al. 2019). DGL provides an interface for implementing operations on graphs, such as the message-passing framework used in many GNN architectures.

The traditional methods are already implemented in NetworkX 2.5 (Hagberg et al. 2008), a Python library containing many graph algorithms.

Model	Scorer	LR	LR schedule	aggregate
GCN	DistMult	$6 \cdot 10^{-4}$	`[20]`	sum

Table 6.7: Specific hyperparameters for the favourite POS prediction task.

Parameter	Value
epochs	4
batch size	128
learning rate	10^{-5}
encoder layers dimensionality	`[29, 128]`
decoder layers dimensionality	`[128, 29]`
KL-divergence weight	0.9

Table 6.8: VAE hyperparameters. See Section 6.5.1 for more information

Most GNN models we use are implemented in DGL.

GCN, GraphSAGE, and GAT could be used as they are with our experimental settings. We adjusted the GIN model so that it works with mini-batch training. This can be easily done by following the DGL documentation (DGL 2020).

All of the models work only with simple graphs (without multiple types of nodes and edges). To overcome this, we use `dgl.nn.HeteroGraphConv`, a class in DGL, to which a GNN model is supplied for each type of edge. We use the same model architecture for all edge types (with independent, trainable parameters).

We work with Python 3.8.3.

6.4 Dataset CU

In order to reduce the computational requirements, we first reduce the dataset to only "active" nodes and edges between them. We believe that it is desirable to experiment with as many methods and as many hyperparameters as possible. Although running some of these experiments for the whole dataset would be feasible, it would make the process considerably slower. As the objective of this thesis is not to train the best possible model on the whole data but rather compare selected methods, we chose to limit the dataset to speed up our experiments. For example, pretraining of a GCN model on a selected subset of edges takes more than a day on the secured server, and we need to run it several times to test more than one set of hyperparameters. Such process would not be possible on a dataset order of magnitude larger, as can be seen in Table 6.1. We write about the need to reduce in dataset in detail in Section 6.1.

6.4.1 Subsets of Data

Initially, we started with a dataset consisting of only one type of edges between nodes of the same type. The client–client similarity edges are aggregated into one, and the new edge weight is the sum of the weights of the original edges between given nodes. The resulting graph (containing only one similarity edge between a pair of clients) is filtered so that each person has at most 30 neighbours.

Edge Type	Edge Variant	Edges Used
Client - Client	*aggregated*	top 30
Client - Client	Friends	all
Client - Client	Household	all
Client - ATM	Withdrawing from an ATM	all
Client - Counterparty	Receiving money	all
Client - Counterparty	Sending money	all
Client - Counterparty	Alignment	all
Client - Employer	Receiving money	all
Client - MCC	Shopping in shop with MCC	top 10
Client - Shop	Shopping in an eshop	all
Client - Store	Shopping in a store	all
Client - ZIP	Client ZIP	all

Table 6.9: Edges in the dataset D-Sim. Column Edges Used denotes whether the original data have been restricted, "top 10" means that for each node, edges to the 10 most similar nodes (of given type) are kept.

The number (30) of neighbours is selected so that the dataset is small enough to allow reasonable training time and RAM consumption while still retaining enough information about the network. The top 10 of most frequented MCCs define the favourite MCCs for each client. All of the edges we aim to predict are also selected from the database. The used edges are described in Table 6.9. We call this dataset **D-Sim**.

Then, instead of using aggregated similarity edges, we use each similarity edge type separately and limit them so that each client has a similarity edge of the given type to 10 of its neighbours. The number (10) is selected so that the network is small enough. The used edges are described in Table 6.10. This dataset cannot be used with non-GNN methods as is because there are multiple types of edges between nodes of the same types. When we use non-GNN methods (Jaccard coefficient, Adamic-Adar, etc.) on this data, we merge multiple edges to one. We call this dataset **D-All**.

We also create a dataset consisting of edges used in dataset D-All, except the client–client similarity edges. We do this because it would be desirable to know whether (and how much) the link prediction performance improves by adding the similarity edges. If the link prediction is not significantly better with similarity edges on the input, it means that they might not be used altogether. This would make the system much more scalable, as the similarity edges need not be computed for each pair of (client) nodes. The used edges are described in Table 6.11. We call this dataset **D-NoSim**.

For client nodes we also use attribute features shown in Table 6.12. Ordinal features are normalised, nominal features are one-hot encoded. Except for the education status, which is converted to ordinal feature because it can be linearly ordered. When a feature is missing for a client, we add a new, binary feature for it, indicating whether the original feature is not null.

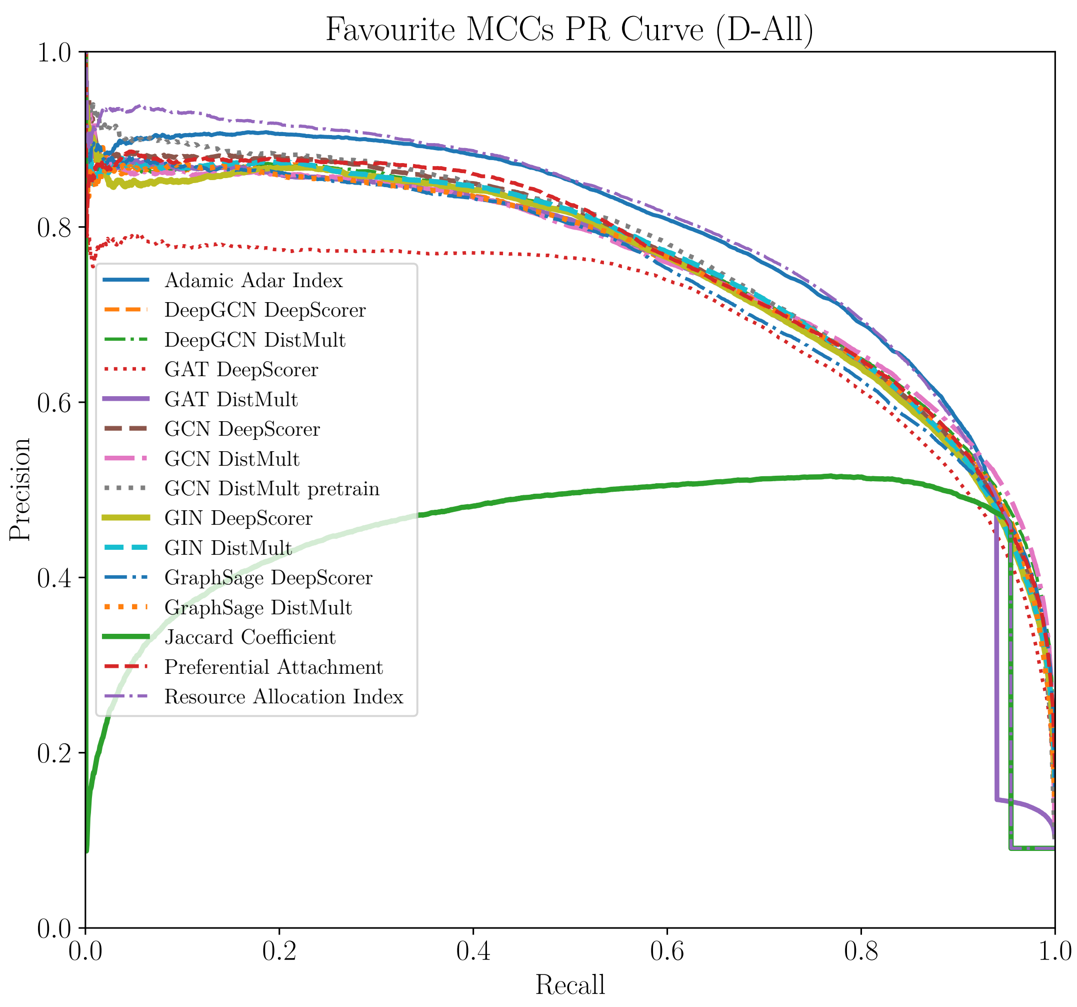

Figure 6.1: Precision-recall curve of favourite MCC prediction on D-All dataset.

6.4.2 Favourite MCC

For our first experiment, we chose to predict favourite MCCs. In the subset of the data we use, there are 305 173 client–MCC edges. As mentioned in Chapter 1, we believe that predicting favourite MCCs might help solving the loan problem.

Using datasets D-All, D-Sim, D-NoSim, we ran approximately 300 experiments on this task, differing in model and hyperparameter selection. Each experiment takes approximately 30 minutes to 4 hours to finish on our secured server. And pretraining the model on the other types of edges takes about one day.

We randomly selected 10% (about 30 000) client–MCC edges and removed them completely from the graph. The training was performed on the rest of the dataset by predicting masked client–MCC edges. During both the training and the evaluation, we use negative sampling, with 10 negative samples per one positive.

Results

The results are provided in Table 6.13. The actual precision-recall curves of models evaluated on the D-All, and D-NoSim dataset are depicted in Figures 6.1 and 6.2, respectively.

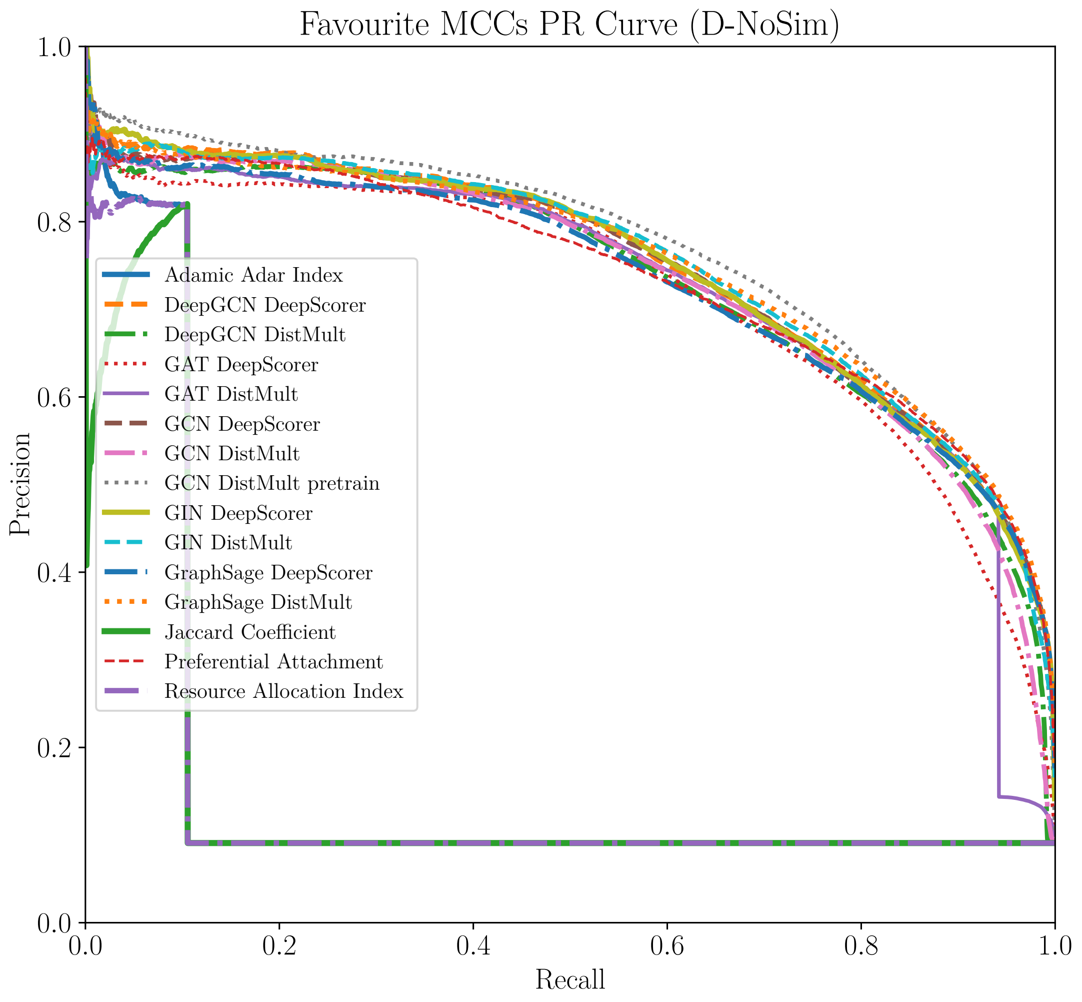

Figure 6.2: Precision-recall curve of favourite MCC prediction on D-NoSim dataset.

For the D-All dataset, we can see in Figure 6.1, that even though some methods perform better than others, the shape of the precision-recall curve is quite similar for most of the models, except the Jaccard Coefficient.

The unusual shape (precision increases as recall increases) of the precision-recall curve of Jaccard Coefficient model is caused by a large number of false positives. More specifically, we believe it is caused by a lot of client–MCC edges with ends that have a large number of incident nodes, and on the other hand, pairs of client and MCC nodes without an edge between them which have small neighbourhoods. This is a direct consequence of the fact that the Adamic Adar Index and the Resource Allocation Index both have high precision at low recall levels and lower precision at high recall levels, and of the Jaccard Coefficient formula 3.1 which instead of the neighbourhood sizes of common neighbours (as in Adamic Adar Index and Recource Allocation Index), it uses the size of the neighbourhood of the end nodes themselves.

Note in Table 6.13 and Figure 6.2, that some methods perform significantly worse on the D-NoSim dataset, namely the Adamic Adar Index, the Jaccard Coefficient, and the Resource Allocation Index. This is caused by a smaller number of common neighbours of nodes between which an edge should be predicted, and these methods mainly depend on the number of common neighbours. On the other hand, the GNN methods perform well as it does not utilise common neighbours, but rather uses learned embeddings for nodes. Preferential Attachment also performs well, even though it uses only neighbourhood sizes of the candidate edges to score them – assuming nodes with large neighbourhoods are likely to link.

We have tried many hyperparameter settings when training the GNN models. The training was quite stable, and almost none of the settings caused a divergence. We believe that the GNN models performed worse than the statistical methods on the D-All dataset because the data is hugely imbalanced. There are 338 MCCs (the most commonly used by clients), and the number of clients which shop at shops with these MCCs is very much imbalanced. On average, 902 people use each MCC but the median is 23, and the top 5% most frequent MCCs are used by more than 3 774 clients each, and the mostly frequented MCC is used by 34 411 clients.

The most used MCCs are 5411 – Grocery Stores, 6011 – Financial Institutions - Manual Cash Disbursements, 5812 – Eating places and Restaurants, and 5912 – Drug Stores and Pharmacies. On the other hand, some airlines, hotels, car rental services have their own MCC, and very few clients have these as their favourite: 3769 – Stratosphere Hotel and Casino, 3513 – Westin Hotels, and 3100 – Malaysian Airline System, are all a favourite MCC of only one client. Because many clients shop in these MCCs, they can probably be predicted with little input from the network structure (intuitively, grocery store are likely to be favourite types of shops of most clients).

6.4.3 Household Members

As previously mentioned in Chapter 1, household member prediction is useful for solving the fraud problem. Specifically, people might lie about not living with another person in order to be given a loan. Banks might like to detect these

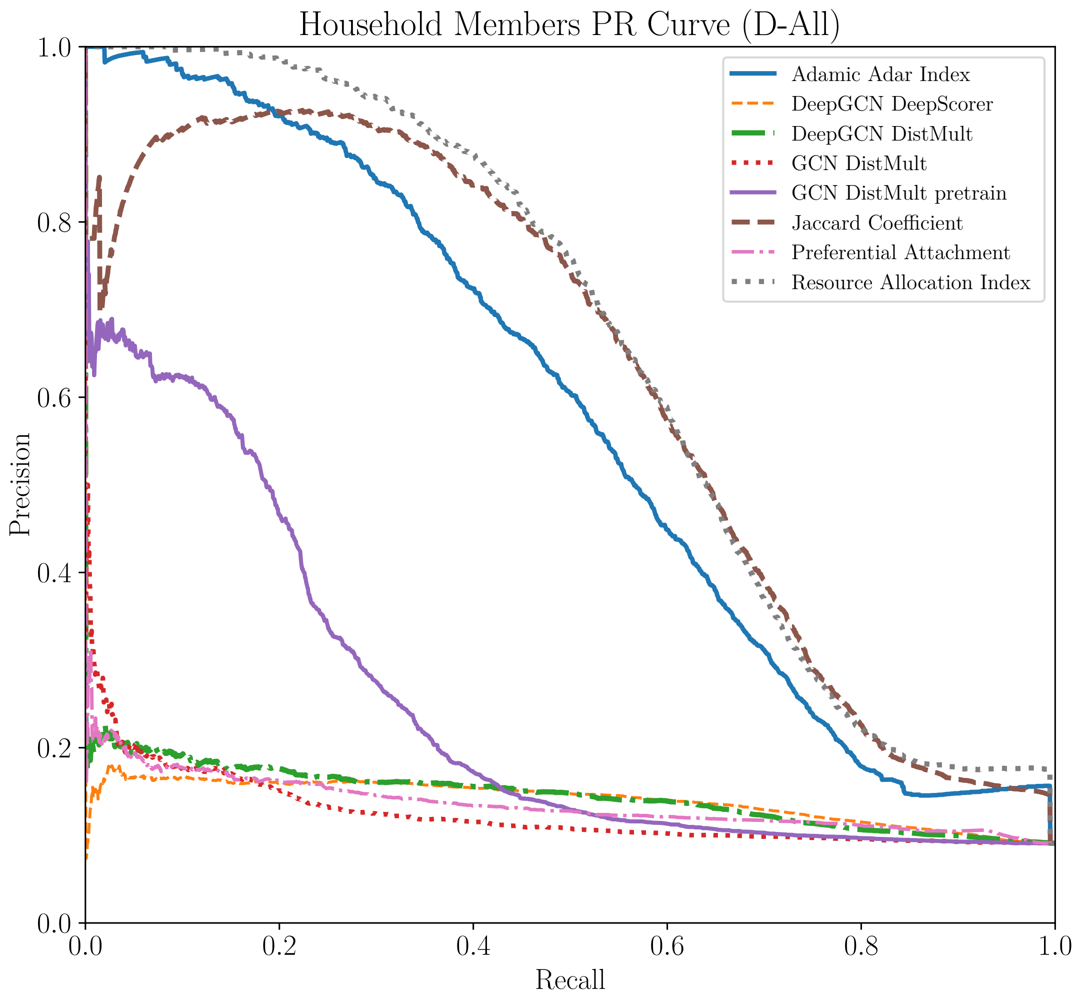

Figure 6.3: Precision-recall curve of household members prediction on D-All dataset.

people.

We try both the traditional methods and GNNs. Because training a GNN model, and especially searching for its hyperparameters, is very time consuming, we chose only the best methods from the previous task, prediction of favourite MCCs.

In the subset of data we use, there are only 3 419 household members edges. We use only 20% of them as training data because we believe that the situation, where less than half of household member edges is known, is more realistic and because there needs to be a reasonable amount of edges for evaluation.

Results

In Table 6.14 and Figures 6.3 and 6.4, we can see that the traditional methods significantly outperform any GNN. The training of GNNs on this task proved challenging. When training to only predict household members, the neural network was very unstable and we were not able to make it converge.

We can see in Figure 6.3, that the pretraining of GCN significantly improves over the non-pretrained model. Unfortunately, the same does not hold

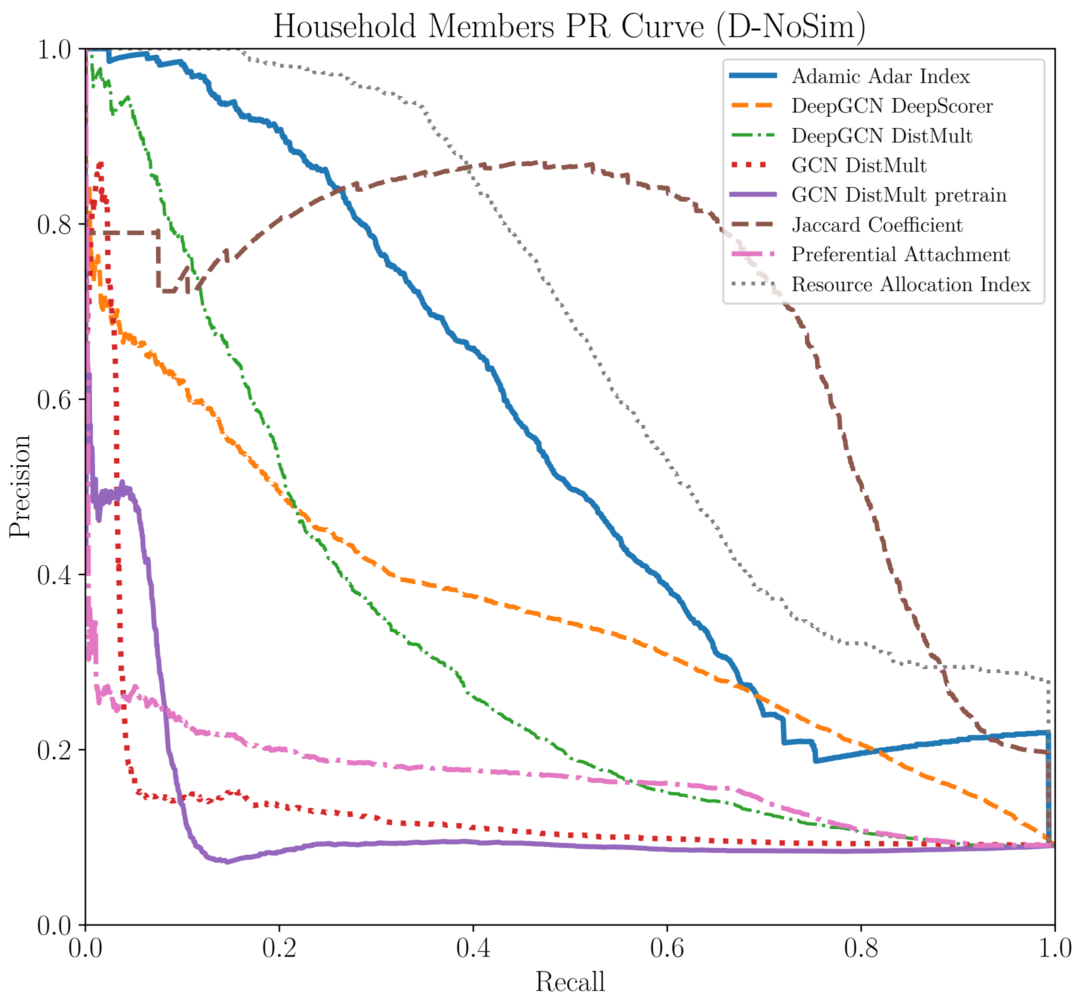

Figure 6.4: Precision-recall curve of household members prediction on D-NoSim dataset.

the D-NoSim dataset, and we observe a negative transfer to the final task. Such behaviour is observed by W. Hu et al. 2020, who show that on some datasets pretraining hurts the performance on the downstream task.

6.4.4 Favourite Shops

Predicting favourite shops of a client is useful for studying their behaviour. For example, it would be useful to know that some clients buy airplane tickets several times a year, these clients may be, for example, offered a travel insurance. It might also be beneficial for predicting whom to offer a loan or a mortgage, e.g. a client who usually shops in stores selling expensive items is likely not to need a loan, but he might need a mortgage at some point.

In the restricted version of our dataset, there are 2 840 053 client–store edges; we use 90% of them for training.

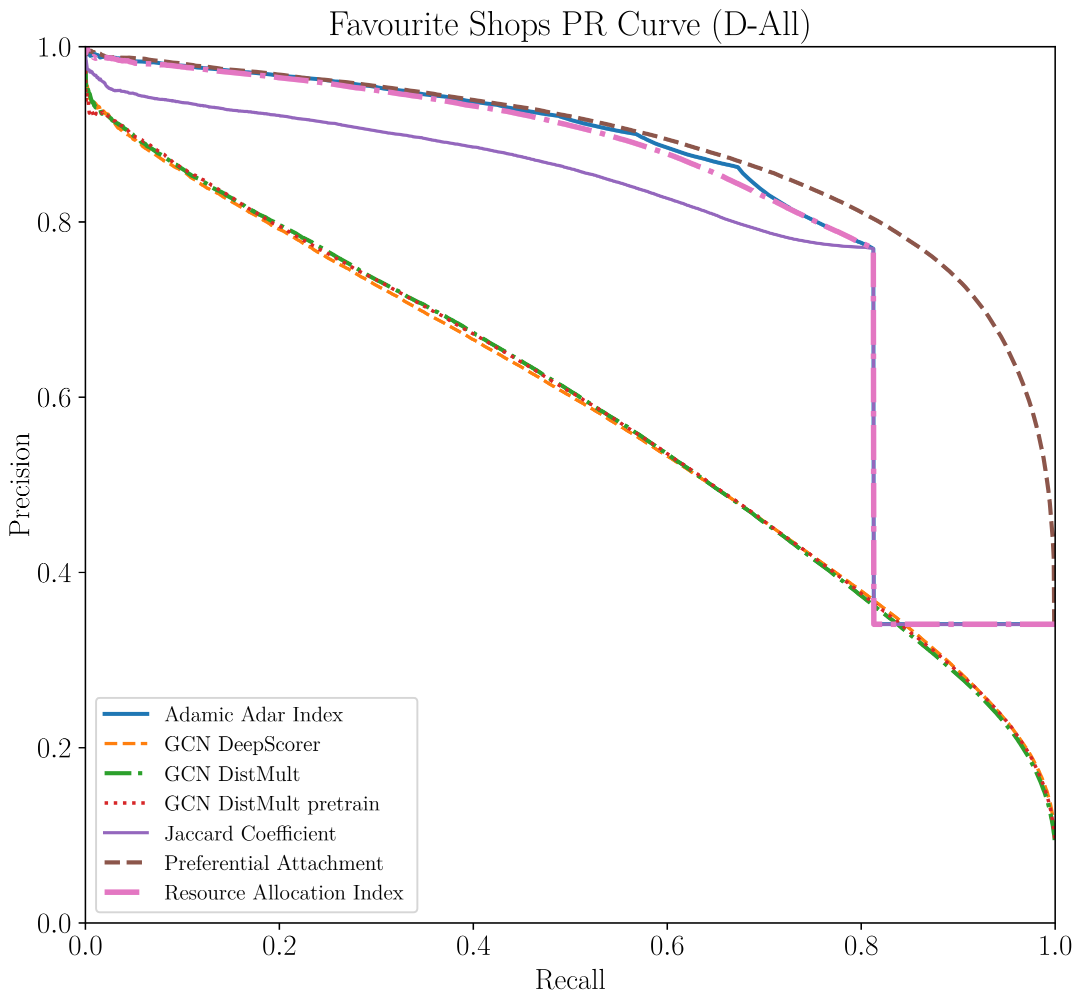

Figure 6.5: Precision-recall curve of favourite stores prediction on D-All dataset.

Results

See Table 6.15 and Figures 6.5 and 6.6 for the results. We tried to use the DeepGCN model, but after a few epochs the training failed due to a memory error. This behaviour is strange because the memory usage should not be increasing dramatically after every epoch; we suspect there is an error in DGL, the graph neural network library we use.

On both the D-All, and the D-NoSim dataset there is a clear drop in precision of Adamic-Adar Index, Jaccard Coefficient, and Resource Allocation Index. This is caused by edges with end nodes that do not have many common neighbours, as we explained in Section 6.4.2.

Note that all the GNN methods have relatively similar AP. It is caused by an unbalanced distribution of favourite shops among clients. On average, a store is a favourite of 15 clients. However, the top 1% of shops is frequented by more than 1 195 clients. 11 678 clients frequently shop in the most favourite store: McDonald's Prague. Ikea is the second most favourite store with 10 381 clients, and Tesco in Nový Smíchov, Prague is the third most frequented store with 10 010 clients. See Figure 6.7 for the distribution.

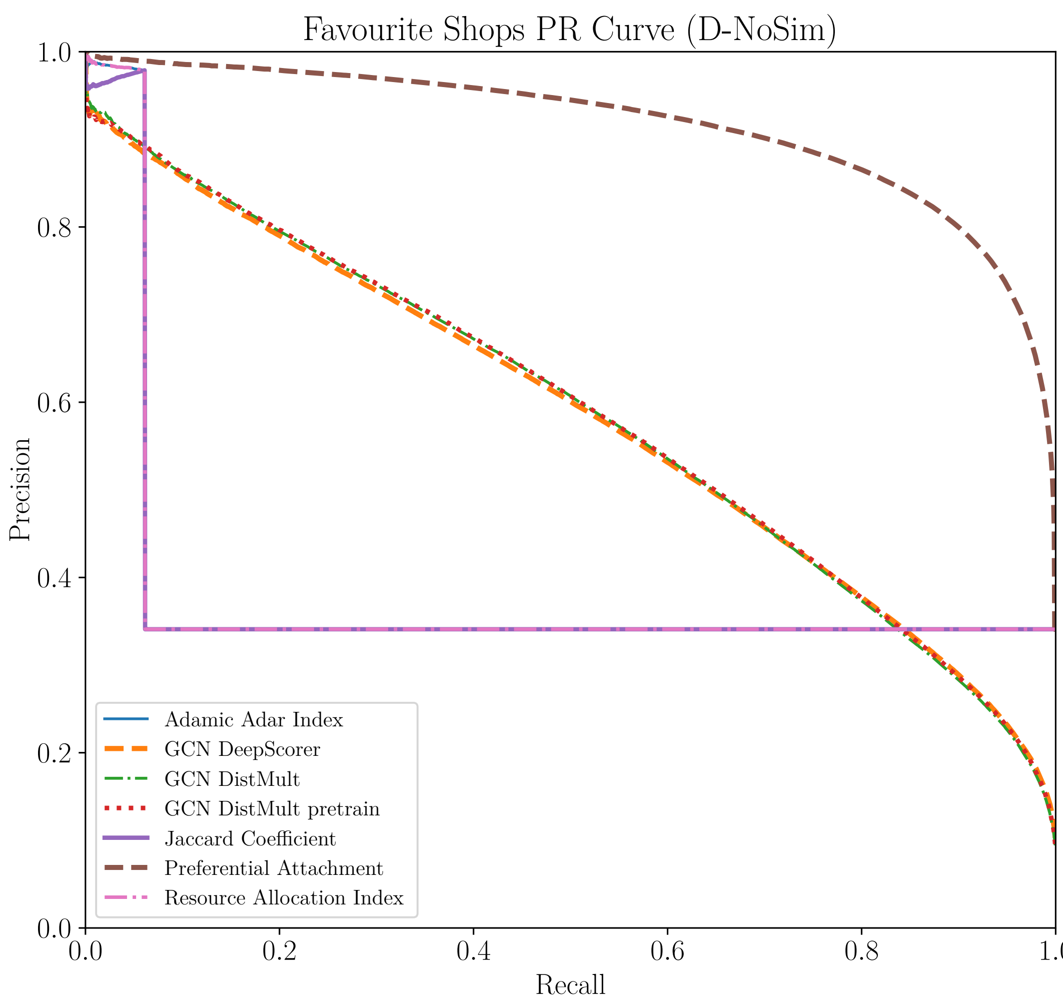

Figure 6.6: Precision-recall curve of favourite stores prediction on D-NoSim dataset.

6.4.5 Friends

Predicting friendship relations among clients can be beneficial for many tasks. Knowing which clients are friends enables the bank to more efficiently market its products, because friends might share information about their banking product. A better product, e.g., a mortgage with a lower interest rate, might be offered to a client who has many friends who are in a situation in which they might also need a mortgage.

Also, clients whose friends are defaulting on their loans may be more likely to default, because their friends may have the same employer, or they live in the same area where the employment heavily depends on a particular industry, e.g., coal mines.

In the restricted dataset, there are 2 877 client–client friendship edges.

Results

As can be seen in Table 6.16, statistical methods performed very well. From that,test it is obvious that friendship links are between nodes that already have a lot of common neighbours. Interestingly, the performance is often better when

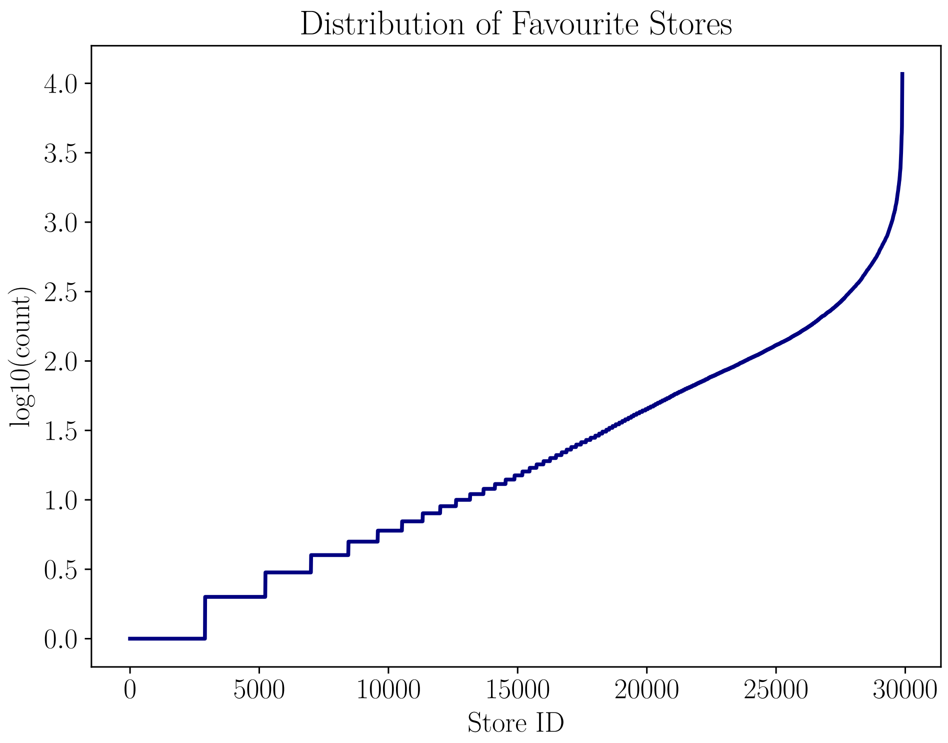

Figure 6.7: Distribution of favourite shops among clients. The vertical axis denotes the logarithm of the number of clients which have given shops as favourite.

the method is used on a dataset that does not use any similarity edges between clients. That can actually be viewed as an advantage, as the similarity edges need not be precomputed in order to predict friends.

Some of the statistical methods perform very well, such as the Resource Allocation Index with 0.92 AP. It is a consequence of how these edges were inferred – clients who often frequented the same restaurants at similar times are considered friends. This creates a short path in the network between the clients – both using similarity of the clients and by using the client–store payments edge. This is precisely why statistical methods using common neighbours work well on this task.

The labelled dataset (edges to be predicted) size is very small for a neural network, it is hence difficult to make the GNN to generalise.

We suspect that DeepScorer performs better than DistMult because it is more “powerful” (it has more parameters, it is deeper), hence being able to learn better final embedding and not depending so heavily on the GNN architecture and the network structure. The same holds for DeepGCN.

6.5 Dataset PF

As this is a dynamic dataset; we predict the edges in the future. Given client similarity edges and their favourite POSs in one month, we predict which POSs will be their favourite in the next month. Even though we only predict one month

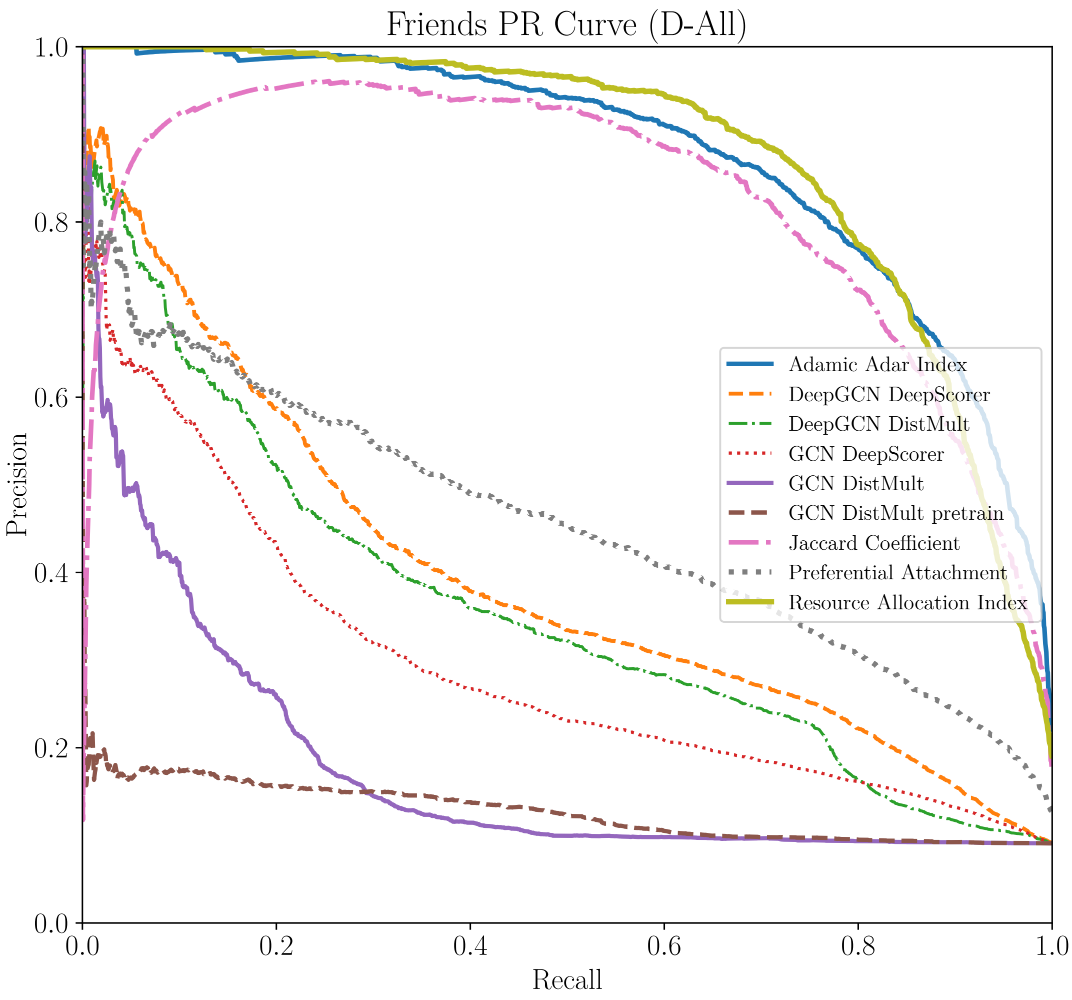

Figure 6.8: Precision-recall curve of friends prediction on D-All dataset.

to the future, we use only one model for all months. An autoregressive approach could be used to predict the edges in more distant future: at month t infer edges for month $t+1$, and using the inferred edges, predict edges for month $t+2$, etc.

The data has been gathered in a two-year period; we set aside 4 months for evaluation (4 months to be predicted, so the data actually comes from 5 months). We restricted the client similarity edges so that each client has at most 20 neighbours per each similarity edge type. The number of neighbours is selected so that the training of the models is reasonably fast. Even after this, the training of GNNs takes approximately 20 hours until convergence.

6.5.1 POS information

As described in Section 4.2.2, the dimensionality of POS features is too high. Were the features continuous, we would have used some traditional method for dimensionality reduction, such as Principle Component Analysis (Pearson 1901). However, since the features are discrete, and the dimensionality is quite high with respect to the number of examples, we need to use a more sophisticated dimensionality reduction method. We chose Variational Autoencoder (VAE) (Kingma et al. 2014).

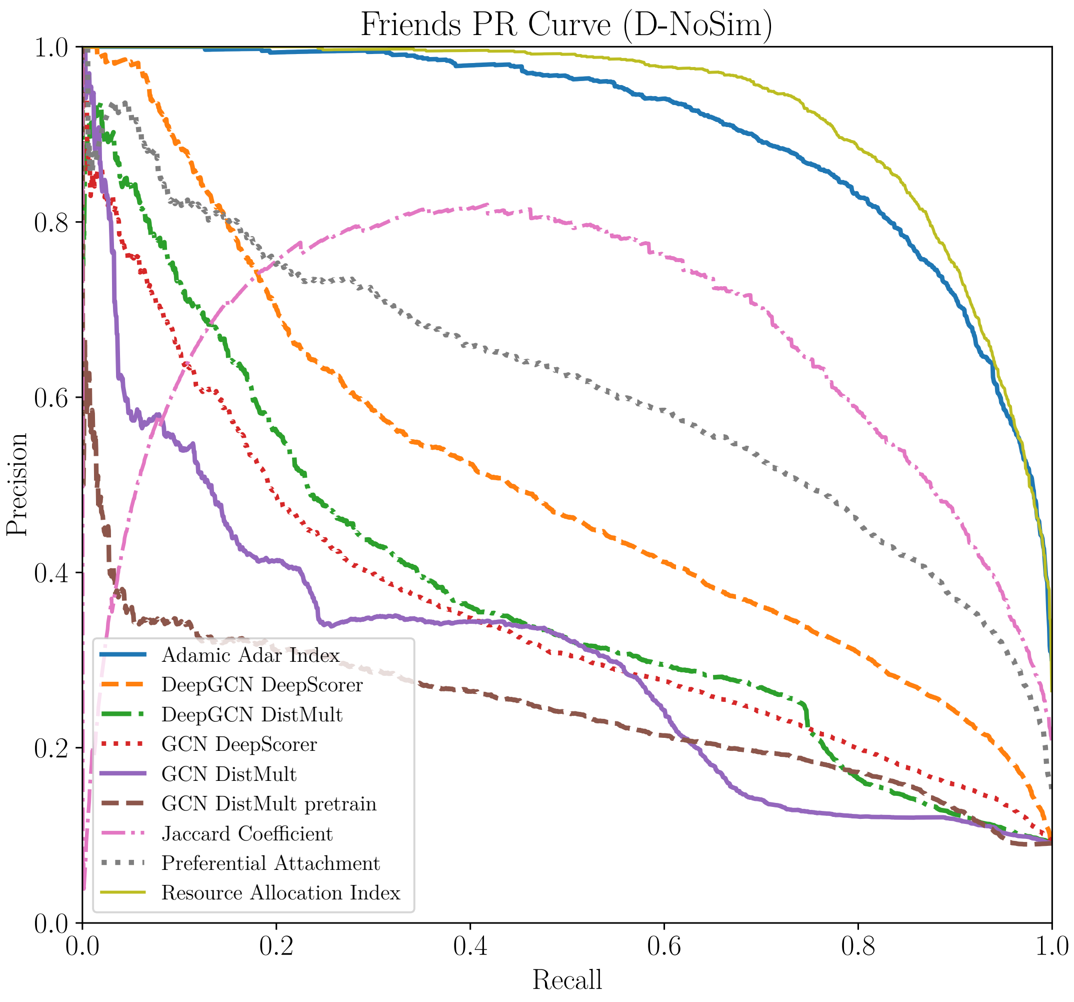

Figure 6.9: Precision-recall curve of friends prediction on D-NoSim dataset.

VAE is a neural network model which learns a mapping from the features space to a latent, less dimensional space (encoder), and mapping from the latent space back to the feature space (decoder). The *variational* component of VAE, represented as part of the loss function, ensures that the examples are not mapped to a discrete point in the latent space but are mapped to a ball in the latent space. Even though VAE is much more prominently used for generating new, artificial examples (often pictures), it can be used as a dimensionality reduction technique.

After training VAE, the low dimensional representation is obtained by mapping the POSs using the encoder to the latent space. We use a 29 dimensional latent space (same as the features of client nodes).

See Table 6.8 for the list of parameters used during training of VAE.

6.5.2 Results

In Table 6.17, we report the average precision of next month's favourite POS prediction. In this task, a GNN achieves significantly better results than most methods. It is caused by the property of the Adamic Adar Index, the Jaccard Coefficient, and the Resource allocation Index, giving nodes that do not share any common neighbours zero scores. More clearly, this can be seen in Figure 6.10,

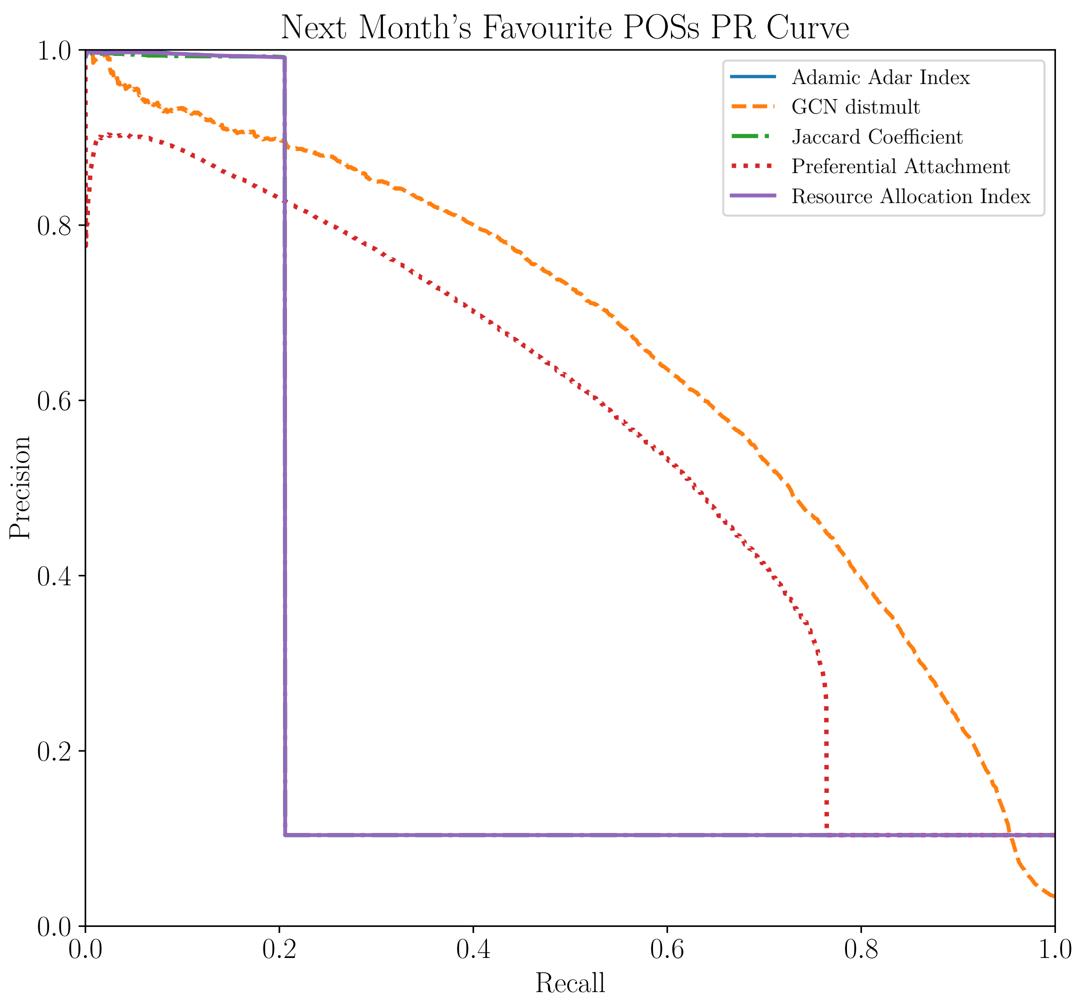

Figure 6.10: Precision-recall curve of next month favourite POS prediction.

where there is a sudden drop in precision, around 0.2 recall level.

This task is harder than those from Section 6.4, because the edges we are predicting were not inferred from the rest of the data on which the prediction is conditioned. It is quite interesting to see that Preferential Attachment has reasonably good results because this method only uses the number of neighbours (not common neighbours) for each node when predicting a link.

6.6 Failed Attempts

Initially, after reading GNN literature described in Chapters 2 and 3, we set out to implement those methods and experiment on our datasets. However, it turned out to be quite challenging.

Firstly, using a GNN on a line graph as in the paper by Cai et al. 2020 is impossible with a dataset as large as ours. The number of vertices in a line graph is the same as the number of edges in the original graph, by definition, and the number of edges of a line graph is much higher than the original number of vertices (especially if the average vertex degree is high). The largest graph the original paper uses is about 7 thousand nodes and 50 thousand edges, whereas we have hundreds of thousands of vertices and millions of edges.

We also wanted to fully implement RGCN, including its matrix decomposition. However, it has been a challenge to implement RGCN under our experimentation settings (mini batch training) in the framework of our choice: DGL. Nevertheless, using GCN on our graph is very similar to RGCN, as there are not many edge types, and the weight matrix decomposition used in RGCN would not help (as stated by the authors).

Edge Type	Edge Variant	Edges used
Client - Client	Outgoing payments	top 10
Client - Client	Outgoing domestic payments	top 10
Client - Client	Outgoing foreign payments	top 10
Client - Client	Incoming payments	top 10
Client - Client	Incoming domestic payments	top 10
Client - Client	Incoming foreign payments	top 10
Client - Client	Salary transfer	top 10
Client - Client	SIPO payments	top 10
Client - Client	Permanent payments	top 10
Client - Client	Loan receiving	top 10
Client - Client	Loan instalements	top 10
Client - Client	Debit card transactions	top 10
Client - Client	Debit card payment	top 10
Client - Client	Debit card payment with cashback	top 10
Client - Client	ATM withdrawal	top 10
Client - Client	ATM withdrawal domestic	top 10
Client - Client	ATM withdrawal foreign	top 10
Client - Client	MCC class	top 10
Client - Client	Friends	all
Client - Client	Household	all
Client - ATM	Withdrawing from an ATM	all
Client - Counterparty	Receiving money	all
Client - Counterparty	Sending money	all
Client - Counterparty	Alignment	all
Client - Employer	Receiving money	all
Client - MCC	Shopping in shop with MCC	top 10
Client - Shop	Shopping in an eshop	all
Client - Store	Shopping in a store	all
Client - ZIP	Client ZIP	all

Table 6.10: Edges in the dataset D-All. Column “Edges used” denotes whether the original data have been restricted, “top 10” means that for each node, edges to the 10 most similar nodes (of given type) are kept.

Edge Type	Edge Variant	Edges used
Client - Client	Friends	all
Client - Client	Household	all
Client - ATM	Withdrawing from an ATM	all
Client - Counterparty	Receiving money	all
Client - Counterparty	Sending money	all
Client - Counterparty	Alignment	all
Client - Employer	Receiving money	all
Client - MCC	Shopping in shop with MCC	top 10
Client - Shop	Shopping in an eshop	all
Client - Store	Shopping in a store	all
Client - ZIP	Client ZIP	all

Table 6.11: Edges in the dataset D-NoSim. Column "Edges used" denotes whether the original data have been restricted, "top 10" means that for each node, edges to the 10 most similar nodes (of given type) are kept.

Feature	Ordinal	Cardinality	One-hot Encoded
Is physical person	no	2	no
Birth year	yes	—	no
Gender	no	2	no
Score	yes	—	no
Education	no	8	no
Social status	no	7	yes
Marital status	no	5	yes
Number of household member	yes	—	no
Number of children	yes	—	no

Table 6.12: Client features.

Method	Scorer	Pretrain	Data	AP
Adamic Adar Index	—	—	D-All	0.7763
Adamic Adar Index	—	—	D-Sim	0.7771
Adamic Adar Index	—	—	D-NoSim	0.1700
Jaccard Coefficient	—	—	D-All	0.5376
Jaccard Coefficient	—	—	D-Sim	0.4422
Jaccard Coefficient	—	—	D-NoSim	0.1580
Resource Allocation Index	—	—	D-All	0.7839
Resource Allocation Index	—	—	D-Sim	0.7865
Resource Allocation Index	—	—	D-NoSim	0.1677
Preferential Attachment	—	—	D-All	0.7674
Preferential Attachment	—	—	D-Sim	0.7610
Preferential Attachment	—	—	D-NoSim	0.7370
GCN	DistMult	yes	D-NoSim	0.7648
GCN	DistMult	yes	D-All	0.7636
GCN	DistMult	no	D-Sim	0.7529
GCN	DistMult	no	D-All	0.7628
GIN	DistMult	no	D-All	0.7552
GraphSAGE	DistMult	no	D-All	0.7462
GAT	DistMult	no	D-All	0.7345
DeepGCN	DistMult	no	D-All	0.7554
GCN	DeepScorer	no	D-All	0.7594
GIN	DeepScorer	no	D-All	0.7467
GraphSAGE	DeepScorer	no	D-All	0.7509
GAT	DeepScorer	no	D-All	0.6959
DeepGCN	DeepScorer	no	D-All	0.7516
GCN	DistMult	no	D-NoSim	0.7611
GIN	DistMult	no	D-NoSim	0.7504
GraphSAGE	DistMult	no	D-NoSim	0.7521
GAT	DistMult	no	D-NoSim	0.7219
DeepGCN	DistMult	no	D-NoSim	0.7308
GCN	DeepScorer	no	D-NoSim	0.7469
GIN	DeepScorer	no	D-NoSim	0.7482
GraphSAGE	DeepScorer	no	D-NoSim	0.7333
GAT	DeepScorer	no	D-NoSim	0.7168
DeepGCN	DeepScorer	no	D-NoSim	0.7495

Table 6.13: MCC prediction scores with 90% training data.

Method	Scorer	Pretrain	Data	AP
Adamic Adar Index	—	—	D-All	0.5739
Adamic Adar Index	—	—	D-NoSim	0.5441
Jaccard Coefficient	—	—	D-All	0.6143
Jaccard Coefficient	—	—	D-NoSim	0.6954
Resource Allocation Index	—	—	D-All	0.6482
Resource Allocation Index	—	—	D-NoSim	0.6636
Preferential Attachment	—	—	D-All	0.1355
Preferential Attachment	—	—	D-NoSim	0.1665
GCN	DistMult	yes	D-NoSim	0.1197
GCN	DistMult	no	D-NoSim	0.1335
GCN	DistMult	no	D-All	0.1245
GCN	DistMult	yes	D-All	0.2449
DeepGCN	DistMult	no	D-NoSim	0.3131
DeepGCN	DistMult	no	D-All	0.1443
DeepGCN	DeepScorer	no	D-All	0.1399
DeepGCN	DeepScorer	no	D-NoSim	0.3578

Table 6.14: Household members prediction with 20% training data.

Method	Scorer	Pretrain	Data	AP
Adamic Adar Index	—	—	D-All	0.8100
Adamic Adar Index	—	—	D-NoSim	0.3803
Jaccard Coefficient	—	—	D-All	0.7726
Jaccard Coefficient	—	—	D-NoSim	0.3794
Resource Allocation Index	—	—	D-All	0.8064
Resource Allocation Index	—	—	D-NoSim	0.3803
Preferential Attachment	—	—	D-All	0.8836
Preferential Attachment	—	—	D-NoSim	0.9144
GCN	DistMult	no	D-All	0.5869
GCN	DistMult	no	D-NoSim	0.5867
GCN	DeepScorer	no	D-All	0.5852
GCN	DeepScorer	no	D-NoSim	0.5842
GCN	DistMult	yes	D-All	0.5875
GCN	DistMult	yes	D-NoSim	0.5882

Table 6.15: Favourite shops prediction with 90% training data.

Method	Scorer	Pretrain	Data	AP
Adamic Adar Index	—	—	D-All	0.8736
Adamic Adar Index	—	—	D-NoSim	0.9046
Jaccard Coefficient	—	—	D-All	0.8193
Jaccard Coefficient	—	—	D-NoSim	0.6574
Resource Allocation Index	—	—	D-All	0.8754
Resource Allocation Index	—	—	D-NoSim	0.9290
Preferential Attachment	—	—	D-All	0.4552
Preferential Attachment	—	—	D-NoSim	0.6132
GCN	DistMult	no	D-NoSim	0.3023
GCN	DistMult	yes	D-NoSim	0.2760
GCN	DistMult	yes	D-All	0.1294
GCN	DistMult	no	D-All	0.1747
GCN	DeepScorer	no	D-All	0.2928
GCN	DeepScorer	no	D-NoSim	0.3562
DeepGCN	DeepScorer	no	D-All	0.3927
DeepGCN	DeepScorer	no	D-NoSim	0.5058
DeepGCN	DistMult	no	D-All	0.3575
DeepGCN	DistMult	no	D-NoSim	0.3774

Table 6.16: Friends prediction with 20% training data.

Method	Scorer	AP
Adamic Adar Index	—	0.2750
Jaccard Coefficient	—	0.2745
Resource Allocation Index	—	0.2749
Preferential Attachment	—	0.5647
GCN	DistMult	0.6659

Table 6.17: Prediction of next month's favourite POSs.

Conclusion

This thesis focuses on link prediction methods and their usage in the context of inferred social networks. Our goal was to evaluate verified link prediction methods on a real world ISN.

We reviewed common link prediction methods and state-of-the-art GNNs architectures and approaches for training them. Selected GNNs were then applied to our datasets and used for solving several link prediction tasks. We show that it is difficult to outperform traditional and relatively simple link prediction methods, such as Adamic Adar Index, with complex neural networks, even after an extensive hyperparameter search. We also show that GNNs achieve higher precision than the verified link prediction methods when the tasks are harder, and the existence of a predicted link is likely even between nodes sharing no common neighbours, which happens more often when the edges are not inferred from the data, on which the GNN is conditioned.

We proposed an adjustment to an existing GCN architecture, called DeepGCN, and an alternative to DistMult, a link scoring method, called DeepScorer. Both DeepGCN and DeepScorer are comparable to GCN and DistMult, respectively, across a variety of tasks. In some cases, the DeepGCN outperforms the original GCN, especially when the dataset is smaller.

Overall, GNNs are difficult to optimise. Most methods were originally proposed for a much smaller dataset or datasets consisting of a large number of small graphs. Nevertheless, we have shown that some GNN methods can be used even on relatively large datasets. On the other hand, we have found out that GNNs tend to overfit on very small training data, and our model (DeepGCN), is able to generalise better on these datasets.

We have experimented with GNN pretraining to address the issues with a small training dataset. A pretrained GNN model achieves relatively high scores on some small datasets but still underperforms the traditional link prediction methods. On one task, the pretraining caused a negative transfer, very much underperforming the other models.

Using a good ISN is crucial for achieving the best results. We show that even simple methods outperform state-of-the-art neural networks when the dataset contains important similarity edges. On the other hand, when the inference of similarity edges between very large sets of nodes is not feasible, GNNs can achieve better results.

There is a clear scalability issue in GNNs, both in terms of time needed to train the network and memory resources required to make the training even feasible. In future work, we believe this issue should be addressed. Also, filling in missing values in node attributes using inferred social network is an interesting topic for future work.

Bibliography

Adamic, Lada A and Eytan Adar (2003). “Friends and Neighbors on the Web”. In: *Social Networks* 25.3, pp. 211–230. ISSN: 03788733. DOI: 10.1016/S0378-8733(03)00009-1.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2016). “Neural Machine Translation by Jointly Learning to Align and Translate”. In: arXiv: 1409.0473.

Bringmann, Bjoern et al. (2010). “Learning and Predicting the Evolution of Social Networks”. In: *IEEE Intelligent Systems* 25.4, pp. 26–35. ISSN: 1541-1672. DOI: 10.1109/MIS.2010.91.

Brugere, Ivan, Brian Gallagher, and Tanya Y. Berger-Wolf (2016). “Network Structure Inference, A Survey: Motivations, Methods, and Applications”. In: arXiv: 1610.00782.

Cai, Lei et al. (2020). “Line Graph Neural Networks for Link Prediction”. In: arXiv: 2010.10046.

Česká Spořitelna (2020). *Who We Are*. URL: https://www.csas.cz/en/about-us/who-we-are (visited on 04/17/2021).

Chakrabarti, Deepayan and Christos Faloutsos (2006). “Graph Mining: Laws, Generators, and Algorithms”. In: *ACM Computing Surveys* 38.1, 2–es. ISSN: 03600300. DOI: 10.1145/1132952.1132954.

Chiang, Wei-Lin et al. (2019). “Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266. DOI: 10.1145/3292500.3330925.

DGL (2020). *6.3 Training GNN for Link Prediction with Neighborhood Sampling*. URL: https://docs.dgl.ai/en/0.5.x/guide/minibatch-link.html (visited on 04/20/2021).

Dwivedi, Vijay Prakash et al. (2020). “Benchmarking Graph Neural Networks”. In: arXiv: 2003.00982.

Fey, Matthias and Jan E. Lenssen (2019). “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fournier-Viger, Philippe et al. (2020). “A Survey of Pattern Mining in Dynamic Graphs”. In: *WIREs Data Mining and Knowledge Discovery* 10.6. ISSN: 1942-4787, 1942-4795. DOI: 10.1002/widm.1372.

Gilmer, Justin et al. (2017). “Neural Message Passing for Quantum Chemistry”. In: arXiv: 1704.01212.

Glorot, Xavier and Yoshua Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Vol. 9. Proceedings of Machine Learning Research. PMLR, pp. 249–256.

Goldschmidt, O. and D.S. Hochbaum (1988). “Polynomial Algorithm for the K-Cut Problem”. In: *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*. White Plains, NY, USA: IEEE, pp. 444–451. ISBN: 978-0-8186-0877-3. DOI: 10.1109/SFCS.1988.21960.

Grattarola, Daniele and Cesare Alippi (2020). “Graph Neural Networks in TensorFlow and Keras with Spektral”. In: arXiv: 2006.12138.

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart (2008). “Exploring Network Structure, Dynamics, and Function using NetworkX”. In: *Proceedings of the 7th Python in Science Conference.* Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, pp. 11–15.

Hamilton, William L., Rex Ying, and Jure Leskovec (2018). “Inductive Representation Learning on Large Graphs”. In: arXiv: 1706.02216.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

Holubova, Irena et al. (2019). “Inferred Social Networks: A Case Study”. In: *2019 International Conference on Data Mining Workshops (ICDMW).* Beijing, China: IEEE, pp. 65–68. ISBN: 978-1-72814-896-0. DOI: 10.1109/ICDMW.2019.00019.

Hu, Weihua et al. (2020). “Strategies for Pre-Training Graph Neural Networks”. In: *International Conference on Learning Representations.*

Hu, Ziniu et al. (2020). “GPT-GNN: Generative Pre-Training of Graph Neural Networks”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* ACM, pp. 1857–1867. ISBN: 978-1-4503-7998-4. DOI: 10.1145/3394486.3403237.

Ivanov, Sergei (2020). *ICLR 2020 Graph Papers.* URL: https://medium.com/@sergei.ivanov_24894/iclr-2020-graph-papers-9bc2e90e56b0 (visited on 10/28/2020).

Jaccard, Paul (1912). “The Distribution of The Flora in The Alpine Zone”. In: *New Phytologist* 11.2, pp. 37–50. ISSN: 0028-646X, 1469-8137. DOI: 10.1111/j.1469-8137.1912.tb05611.x.

Karypis, George and Vipin Kumar (1999). “A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM Journal on Scientific Computing* 20.1, pp. 359–392.

Kingma, Diederik P. and Max Welling (2014). “Auto-Encoding Variational Bayes”. In: arXiv: 1312.6114.

Kipf, Thomas N. and Max Welling (2017). “Semi-Supervised Classification with Graph Convolutional Networks”. In: arXiv: 1609.02907.

Leman, Andrei and Boris Weisfeiler (1968). “The Reduction of a Graph to Canonical Form and the Algebra which appears therein”. In:

Leskovec, Jurij, Anand Rajaraman, and Jeffrey D. Ullman (2020). *Mining of Massive Datasets.* Third edition. New York, NY: Cambridge University Press. ISBN: 978-1-108-47634-8.

Liben-Nowell, David and Jon Kleinberg (2003). “The Link Prediction Problem for Social Networks”. In: p. 19.

Liu, Zhiyuan and Jie Zhou (2020). “Introduction to Graph Neural Networks”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.2, pp. 1–127. DOI: 10.2200/S00980ED1V01Y202001AIM045.

Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng (2013). “Rectifier nonlinearities improve neural network acoustic models”. In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing.*

Mikolov, Tomas et al. (2013). “Distributed Representations of Words and Phrases and Their Compositionality”. In: arXiv: 1310.4546.

Paszke, Adam et al. (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pearson, Karl (1901). “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11, pp. 559–572. DOI: 10.1080/14786440109462720.

Popel, Martin and Ondřej Bojar (2018). “Training Tips for the Transformer Model”. In: *The Prague Bulletin of Mathematical Linguistics* 110.1, pp. 43–70. DOI: 10.2478/pralin-2018-0002.

Radford, Alec et al. (2019). “Language Models Are Unsupervised Multitask Learners”. In:

Rossi, Emanuele et al. (2020). “Temporal Graph Networks for Deep Learning on Dynamic Graphs”. In: arXiv: 2006.10637.

Scarselli, F. et al. (2009). “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20, pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

Schlichtkrull, Michael et al. (2017). “Modeling Relational Data with Graph Convolutional Networks”. In: arXiv: 1703.06103.

Srivastava, Nitish et al. (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958.

Vaswani, Ashish et al. (2017). “Attention Is All You Need”. In: arXiv: 1706.03762.

Veličković, Petar et al. (2018). “Graph Attention Networks”. In: arXiv: 1710.10903.

Wang, Minjie et al. (2020). “Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks”. In: arXiv: 1909.01315.

Wang, Peng et al. (2015). “Link prediction in social networks: the state-of-the-art”. In: *Science China Information Sciences* 58.1, pp. 1–38. ISSN: 1674-733X, 1869-1919. DOI: 10.1007/s11432-014-5237-y.

Wu, Zonghan et al. (2020). “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2020.2978386.

Xu, Keyulu et al. (2018). “How Powerful Are Graph Neural Networks?” In: arXiv: 1810.00826.

Yang, Bishan et al. (2015). “Embedding Entities and Relations for Learning and Inference in Knowledge Bases”. In: arXiv: 1412.6575.

Yu, Tong and Hong Zhu (2020). “Hyper-Parameter Optimization: A Review of Algorithms and Applications”. In: arXiv: 2003.05689.

Zhang, Muhan and Yixin Chen (2018). “Link Prediction Based on Graph Neural Networks”. In: arXiv: 1802.09691.

Zhou, Tao, Linyuan Lu, and Yi-Cheng Zhang (2009). “Predicting Missing Links via Local Information”. In: *The European Physical Journal B* 71.4, pp. 623–630. DOI: 10.1140/EPJB/E2009-00335-8.

List of Figures

List of Tables

List of Abbreviations