FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Pseudosocial network reconstruction using banking transactional data
Student:	Bc. Peter Kolárovec
Supervisor:	Ing. Jakub Marian, M.Sc.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

There are two basic forms of transactions in the payment system: transfer by payment order from account to account and payment by card. Both types of transactions provide a complementary view of actual interpersonal relationships - from account transactions one can identify, for example, a tenant-tenant relationship, from card transactions, for example, friends paying repeatedly at the same time in the same restaurant.

The aim of the thesis is:

1) To learn about the specifics of bank transaction data (time component, amount, transaction notes, banking systems and specifics of data storage).
2) To design an appropriate data/mathematical structure to describe real social relationships (family, colleague, tenant, etc.).
3) Propose methods and algorithms to reconstruct a network of real social ties from transactional data.
4) Implement the proposed methods and algorithms in the environment of Česká spořitelna, a.s. within the student's work placement.
5) Verify the implementation on real data and evaluate the results.

Electronically approved by Ing. Karel Klouda, Ph.D. on 1 April 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Pseudosocial network reconstruction using banking transactional data

Bc. Peter Kolárovec

Department of Applied Mathematics
Supervisor: Ing. Jakub Marian, M.Sc.

June 21, 2022

Acknowledgements

I thank my supervisor Ing. Jakub Marian, M.Sc., for his help, thoughtful notes, and suggestions about my work. Also, I want to thank my other colleagues from the data science team at Česká Spořitelna a.s. for their support. My gratitude also goes to my family and closest friends for their love and support that helped me finish this project.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on June 21, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Kolárovec, Peter. *Pseudosocial network reconstruction using banking transactional data*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Popularita grafových neurónových sietí rastie, pretože tradičné prístupy strojového učenia nedokážu pracovať s komplexnými neeuklidovskými údajmi, ako sú grafy. Táto práca uvádza vzťahy, ktoré možno modelovať v bankovom prostredí. Vybrali sme jeden, na ktorý sa zameriavame – zdieľanie domácnosti. V praktickej časti poskytujeme komplexný prístup, ktorý vychádza z predchádzajúceho moderného výskumu učenia z podgrafov, embeddingov a atribútov na predpovedanie hrán. Poskytujeme spôsob, ako do pôvodného konceptu začleniť hranové atribúty, ktoré získavame z transakčných údajov. Vytvorený model sa následne používa na rozšírenie siete domácností o hrany, ktoré neboli zistené pomocou predchádzajúceho heuristického prístupu.

Kľúčové slová grafové neurónové siete, predikcia hrán, SEAL, hranové atribúty

Abstract

The popularity of Graph Neural Networks is on the rise, as traditional machine learning approaches cannot work with complex non-Euclidean data such as graphs. This thesis provides relationships that can be modeled in a banking environment. We picked one we focus on – sharing a household. In the practical part, we provide an end-to-end approach that builds on previous state-of-the-art research of learning from Subgraphs, Embeddings, and Attributes for Link prediction (SEAL). We provide a way of incorporating edge features we extract from transactional data into the original framework. The built model is then used to enhance the household network with links that were not detected with the previous heuristic approach.

Keywords graph neural network, link prediction, SEAL, edge features

Contents

List of Figures

List of Tables

List of Code examples

Introduction

Knowing the customer in an institution such as a bank is a core business need. To know the customers better and extract the underlying information from the available data is the key to a successful business and competitive advantage.

Traditional machine learning approaches help us meet this need and provide more information to our business partners. These techniques are widely spread and accepted in the banking sector. Many internal processes, like credit scoring or prediction of default, would be nearly impossible without them, whether we talk about classic logistic regression or more advanced approaches such as gradient boosting.

However, these techniques have limitations when we want to mine complex non-Euclidean data, such as graphs. Social networks are, by definition, structured as a graph, and applying traditional machine learning algorithms to them would be troublesome, if not impossible. Our need to work with and data-mine such information materialized in looking for plausible algorithms capable of this profound task.

This thesis starts with an introduction to transactional data and information we can use from them as features that could serve as an input to our model. We then look at the types of relationships we can extract from the transactional data. Out of them, we have chosen one type of relationship – sharing a household – that we want to analyze further and model with a neural network in the practical part of this thesis. We also introduce some known techniques capable of working with graph-like data. We have chosen one that suits our case the best and also because we could implement the use of our transactional features to it.

In the practical part, we provide an end-to-end walk-through of understanding this business use case, graph-like data preprocessing, and modeling, followed by an evaluation. We provide the results of this machine learning pipeline on the out-of-sample data. The model we produce is then used in the evaluation phase, where the whole pipeline is re-used, and we produce predic-

tions on data without the ground truth label to identify new links between clients. We incorporate these newly identified links with the old heuristic algorithm outputs and report the change in counts of the households respectively to their sizes.

The author wrote this thesis as part of his work placement at Česká spořitelna, a.s..

Goals

One of the theoretical goals of this thesis is to understand the specifics of transactional data. This data cannot be used directly as the volume of transactions increases daily by hundreds of thousands. The goal is to find a way of aggregating these data and using them as predictive features for a machine learning approach.

However, the theme of relations and social networks induces a use of non-Euclidean graph-like data that cannot be fed into traditional machine learning algorithms. The goal is to research and propose algorithms that would be capable of using graph-like data. The main objective is to implement such a method and verify it on real-world banking data in the environment of Česká spořitelna, a.s..

Chapter 1

Transactional data

The banking transactions are produced by various subsystems that first must carefully process them. Outputs from these systems are further processed and stored in raw format in the data lake. Then, these raw data are processed in an overnight batch and stored in the bank's normalized data warehouse. This makes the data warehouse the only place of the ground truth. We access this data and provide supplementary analyses and enhancement by new columns.

Some of these newly produced columns are also part of the inputs to our algorithm. They are primarily outputs of heuristic algorithms, which are well defined and continually evaluated for quality.

The table of transactions is the fastest-growing part of any banking data warehouse system. It also holds the most information about the underlying behavior of its clients. However, extracting this behavior – in the case of this thesis, their relationships graph – solely from transactional data is unattainable. One of the challenges would be the size of the data. In the bank environment, there are billions of transactions in the table. The volume of the data is in the hundreds of gigabytes. Another challenge would be a missing linkage of relationships between the clients. As in many data mining challenges, the real power is in data aggregation. Considering we want to model client-client relationships, we need to aggregate these client-client interactions from the transactional data.

In this chapter, we will talk about the nature of transactional data and their utilization in the analytical and data mining framework used in this thesis. We will mainly focus on the components of the transaction that would be beneficial to use as features to train the neural network.

1.1 Time component

The first interesting part of any transaction is its time component. There is not much information we can extract from the date and time contained in

the timestamp of a particular transaction.

Considering the aggregation over the two clients between whom the transaction is realized, we can compute various informative features.

Transaction Frequency With a distinct count over the dates in which transactions were realized between two clients divided by the size of the time window – in days – we can obtain a real number between 0 and 1 that signals the frequency in the monitored time window.

Section of the day Utilizing the time unit of the timestamp, we can determine the section of the day the transactions were realized, e.g., before noon, afternoon, or evening. Counting over these sections would get us a number of the transactions in each section of the client-client relationship.

Regularity We can obtain the regularity of a transaction from two sources. The first one is obvious – when a client sets up a standing order. In the case of features, we can track how many of them clients set up between each other. This information is accessible in our data warehouse. The second one is trickier and consists of a separate algorithm that computes a similarity (time and amount) measure between two transactions on, e.g., a weekly or monthly basis. After setting up a cutoff value that we define as sufficient, we can proclaim these transactions as regular. However, we decided this algorithm is out of the scope of this thesis and can serve as future development.

POS co-payments Considering the account transactions and card payments, it could be possible to track if some clients paid at the same point of sale (POS) in short succession. This sounds promising in theory and not that hard to implement – with enough computing power – but it has some problems. Not every POS is connected to the internet and processes the payments online. Some of them handle transactions in batch, and the time component of timestamps is the same for transactions made in different parts of the day, or even all transactions made in one day have the same timestamp. We decided not to use this information. Carefully handling this issue will be a part of future development.

1.2 Amount component

The second principal component of a transaction is the amount. The volume of the transactions in some time window can tell us only so much. We can also incorporate other statistics, such as mean, median, standard deviation, or maximum and minimum.

It might make a difference if there was one transaction of an enormous amount or more in smaller volumes. Therefore, we can provide the algorithm

features representing a volume ranges – bins – and count the number of transactions that belong to these bins, e.g., 0-500, 501-1000, ...

We can also incorporate a feature representing the roundness of the amount. We can track how many transactions in the time window were rounded up to, for example, hundreds.

1.3 Open-text field component

The third central part of every transaction are the transactional notes that – unlike the time and amount components – might not be available for every one of them. With every transaction made, there is a possibility to fill open-text fields. These boxes consist mainly of transaction note for the sender or receiver.

These notes can be processed for a piece of additional information that we can use as features for the algorithm. For example, if a transaction has a note "cinema", we categorize it with a label "culture". This algorithm for categorizing transactions based on transaction notes is out of the scope of this thesis. The algorithm exists, and we only consume the number of transactions belonging to the categories over the provided time window as features.

Chapter 2

Social networks

In this chapter, we would like to talk about social networks and connections of the client that they provide themselves or that we can obtain by data mining.

"A social network is a set of actors, or other entities, and a set or sets of relations defined on them." [3]

The authors of [4] say these relationships can be binary or valued. "Alice follows Bob on Twitter" is a binary relationship, while "Bob liked three tweets of Alice" is a valued relationship. The links can also be symmetric or asymmetric – Facebook friendship needs mutual confirmation, which makes it symmetric, while in the real world, this friendship could be one-sided.

Some types of relationships can also have a unique property of transitivity. For example, if Alice is an ancestor of Bob and we have other information that Bob is the ancestor of Eve, we also need to consider that Alice is the ancestor of Eve. The property of transitivity can be expressed with the graph theory language: every graph component needs to be fully connected.

In the context of this thesis, the actors are clients – represented as nodes. Relations – represented as edges – are certain relationships between two clients. The primary relationship of this thesis – that we predict in the practical part – is sharing a household. Nonetheless, we provide other relationship types that could be extracted and modeled in future development.

2.1 Household

We define a household as group of people living at the same address that share the same surname or that share a bank product. An address is not a unique representation of a household, as one address can have more than one household – the smallest granularity of an address clients provide in the Czech Republic is a house number. The household relationship is binary, symmetric, and transitive.

Without the application of any sophisticated algorithm, the naive households, with a unique identifier representation, are created by grouping the people at the same contact/permanent address with the same surname or a shared bank product. Detected households then serve as training, validation, and testing set for a graph neural network in the practical part explained in chapter 4.

2.2 Family

In some cases, the family relationship could be the same as a household, but it is not a rule. It is also a binary, symmetric and transitive relationship.

We get part of the information from clients themselves. When a parent comes to set up an account for their children, this information is stored in our data warehouse. This is because the parent needs to be a co-owner of their underage children's account. We also store the information of co-owning a checking account between husband and wife, if they decide to set up this type of account.

A somewhat reliable source of family relationships could also be found in transactional notes and personalized names of senders/receivers – (grand)parents, siblings, ... If a person sends a transaction with a note containing "for mom", or "father" we detect this information and store it in our data warehouse. Notes containing phrases like "pocket money", or "allowance" signal a parent-children relationship.

2.3 Tenant

The tenant-renter relationship can be obtained mainly from transaction data utilizing the algorithm mentioned above to mine the transactional notes. This relationship can provide additional information to the household relationship as the housing unit can have more tenants that do not make transactions with each other but send rent to the same person. This relationship is binary, asymmetric, and non-transitive.

2.4 Coworkers

If we can detect the client's salary through transactional data mining, it could help us detect a coworker relationship between two clients. Usually, these two individuals would receive their salary from the same account. We would first detect the employer-employee relationship and then deduce the coworker relationship.

Further analysis, e.g., POS co-payments, from section 1.1, could strengthen this relationship, as the immediate coworkers might pay for lunch shortly after each other, see example in table 2.1.

client_id	pos_id	timestamp	amount
1	100	20-04-2022 11:48:52	170
2	100	20-04-2022 11:50:02	180
3	200	20-04-2022 11:51:10	175

Table 2.1: Table of possible POS co-payments

This relationship is binary, symmetric, and usually, it is also transitive. There might be some cases if an individual has more sources of income – jobs – where he can have more than one distinct group of coworkers.

2.5 Acquaintances

In some way, we can consider this type of relationship as default – meaning that if two clients do not have any of the above relationships, we can proclaim them as acquaintances if they follow some rules.

Co-payments, mentioned in section 1.1, of two individuals, can be one data source of this information. Repetitive small transactions – paying a restaurant bill or movie tickets – can also signal that these individuals share some social interactions. This relationship is binary. In a real-world application, it is usually asymmetric and non-transitive.

Chapter 3

Representation learning on network data

This machine learning subfield, known as Graph Representation Learning [5], emerged since traditional tabular data cannot provide any more predictive strength, even when used with state-of-the-art algorithms.

However, using complex graph-like data introduces a challenge, as the data is generally unstructured. Researchers have introduced various methods to tackle this problem in the past years, as one can not directly apply the classic machine learning algorithms to this high-dimensional, non-Euclidean data. The typical property of the various methods we will discuss in the subsections below is that they extract the underlying graph structure of the tabular data to yield higher predictive strength.

3.1 Feature engineering

As in any machine learning project, data preparation and feature extraction is the most time-consuming and necessary step of the process, known as CRISP-DM (Cross-industry standard process for data mining).

In addition to the standard tabular features, we can transform raw graph data into a row-based format, using aggregations of the nodes' neighborhoods features or their statistics within the graph [6].

These aggregations and statistics can include:

- Degree of centrality – number of edges that a node x has [7]
- Number of triads – number of distinct induced subgraphs of order 3 [8] that node x is part of
- rooted PageRank algorithm – *"computes a stationary distribution of a random walker starting in node x, who iteratively moves to a random neighbor with probability α or returns with probability $1 - \alpha$"* [9]

3.2 Network embeddings

Network embedding is an unsupervised learning method trying to learn a Euclidean representation of the network in a lower dimension.

The network is represented by an undirected graph $G(V, E)$, where V is a set of nodes and $E \subseteq V \times V$ is a set of edges.

A node embedding is then a function $f : V \mapsto \mathbb{R}^d$, where $d \geq 1$ is the dimensionality of the node embedding [11]. The most used network embedding method is Node2vec [12]. This method learns a mapping – function f – of nodes to a low-dimensional space that maximizes the likelihood of preserving network neighborhoods of the nodes in a latent feature space. The optimization is done by stochastic gradient descent on a graph-based objective function. One of the key features of Node2vec is the use of biased random walks, yielding a trade-off among two basic search methods: breadth-first search (BFS) and depth-first search (DFS) [6].

3.3 Geometric Deep Learning (GDL)

Graph data is structured in a way that can vary significantly from network to network and even node to node of the same network. A graph's support domain is not a uniformly discretized Euclidean space. Therefore the convolution operator – used for signal processing of images or sound – cannot be applied directly to graph-structured data [6].

With this in mind, researchers tried to tackle this issue with Geometric Deep Learning (GDL), a general term for techniques attempting to generalize deep neural models to this non-Euclidean domain [13]. GDL techniques include mainly the use of autoencoders, convolutional and recurrent networks.

3.3.1 Graph Convolutional Networks (GCN)

This type of neural network generalizes the convolution operation so that it can be used with graph-structured data. Using a notation from [14] GCN works on undirected subgraph $G = (X, E)$, extracted around a particular node, which we aim to classify, where:

- X – a node feature matrix of size $N \times d_x$, where N is the number of nodes, d_x is the dimension of the node feature vector
- E – a tensor of edge vector representations of size $N \times N \times d_e$, where d_e is the dimension of edge feature vector

Edges may not have any features, hence the majority of existing graph neural network models perform only a weighted aggregation of node features of the neighbors [14]. In the case of GCN, the aggregation of the features in

the neighborhood is normalized by the number of nodes in it, which is, using notation from [14], defined as:

$$\hat{x_i} = \sigma\left(\frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i} x_j W\right), i = 1, \ldots, N,$$

where $(x_1, \ldots, x_N)$ are node embedding vectors before the convolution operation and $(\hat{x_1}, \ldots, \hat{x_N})$ are their projection after the convolution. W is a matrix of trainable weights, $\mathcal{N}_i$ is a set of immediate neighbors of node i and σ is an activation function (for example Rectified Linear Unit). In [14], they also say that this expression implies that all the neighbors have an equal influence on the chosen node, which is, in most applications, not precisely true.

3.3.2 Graph autoencoders (GAEs)

Autoencoder is an unsupervised method that, in this particular modification, allows obtaining a low dimensional representation of the graph network. Its objective is to reconstruct the original graph structure from the encoded embedding, serving as the low dimensional bottleneck of the original network [6].

In [15], they are using the aforementioned GCN as a building block of the GAE, and with their notation, the embedding matrix $\mathbf{Z}$ (encoder) and the reconstructed adjacency matrix $\hat{\mathbf{A}}$ are calculated as:

$$\mathbf{Z} = GCN\left(\mathbf{X}, \mathbf{A}\right),$$

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^{\intercal}\right)$$

The activation function σ used here is the logistic sigmoid function.

3.4 Link prediction

We can use link prediction as a tool to predict the probability of two nodes from the network forming a connection - edge.

There are various methods to solve this task. Simple yet effective heuristic baseline can be done with extracted features from section 3.1. To compute these heuristics, various sizes of neighborhood are needed. In [9] they define *"h-order heuristic"* as a heuristic which requires knowing up to h-hop neighborhood of the target node. The enclosing subgraph which describes the h-hop neighborhood of given nodes $x, y \in V$ is defined as a subgraph $G^h_{x,y}$ induced from G by the set of nodes $\{i \mid d\left(i, x\right) \leq h \text{ or } d\left(i, y\right) \leq h\}$. Where the function d is the shortest path between two nodes.

For example, common neighbors and the Adamic-Adar index can be classified as first-order and second-order heuristics, respectively. In [9], the heuristics that require knowing the entire network are called high-order heuristics.

For example, rooted PageRank and SimRank are some of those, and they have much better performance. However, too big a subgraph leads to extreme time and memory consumption.

The most significant contribution of [9] is that they show we do not need a large h to learn these high-order features. They have proven that under mild conditions, most high-order heuristics can be effectively approximated from an h-hop enclosing subgraph, where the approximation error decreases at least exponentially with h. This finding enables us to accurately calculate first and second-order heuristics and approximate various high-order heuristics using small enclosing subgraphs around the given link.

3.4.1 Link prediction framework

To tackle the link prediction task, [9] came up with a SEAL (learning from Subgraphs, Embeddings and Attributes for Link prediction) framework. It has three steps:

1. Enclosing subgraph extraction
2. Node information matrix construction
3. GNN (Graph neural network) learning

As previously mentioned in section 3.3, a GNN typically works on a subgraph generally defined in the form of (A, X), where X is the node feature matrix, and A is the adjacency matrix defined as:

- $A_{i,j} = 1$ if $(i, j) \in E$
- $A_{i,j} = 0$ otherwise

The right way of constructing the matrix X is crucial for the successful training of the GNN for link prediction. In the following subsections, we will describe the construction of this matrix shown by [9] with the added possibility of also including edge features, which were not a part of the original framework. The matrix includes node labels, node embeddings, and explicit node features. It will be still of size $N \times d_x$, where $d_x = l + e + f + r$ is the combined length of the feature vectors, where l is the length of the node's label vector, e is the length of the node's embedding vector, f is the length of the node's feature vector, and r is the length of the node's edge feature vector, see figure 3.1 for matrix representation.

3.4.1.1 Node labeling

The artificial component of X is a node label. The idea of this label is to signify the role of a node in an enclosing subgraph. In [9], they defined it in a way that:

$$\begin{bmatrix} nl_{1,1} & \cdots & nl_{1,l} & ne_{1,1} & \cdots & ne_{1,e} & nf_{1,1} & \cdots & nf_{1,f} & ef_{1,1} & \cdots & ef_{1,r} \\ nl_{2,1} & \dots & nl_{2,l} & ne_{2,1} & \dots & ne_{2,e} & nf_{2,1} & \dots & nf_{2,f} & ef_{2,1} & \dots & ef_{2,r} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ nl_{N,1} & \dots & nl_{N,l} & ne_{N,1} & \dots & ne_{N,e} & nf_{N,1} & \dots & nf_{N,f} & ef_{N,1} & \dots & ef_{N,r} \end{bmatrix}$$

Figure 3.1: The conctruction of the matrix X

1. Center nodes are the target nodes, where we predict the occurrence of a link

2. Nodes with various positions relative to the center have different importance to the link

Without such labels, GNN would not be able to tell where the target nodes between which the existence of a link should be predicted are. In [9] they came up with a labeling function $f_l : V \mapsto \mathbb{N}$ which assigns an integer label $f_l(i)$ to every node i in the enclosing subgraph. Two target nodes x and y are always labeled with "1". For every other combination of nodes in the enclosing subgraph they propose a "Double-Radius Node Labeling" (DRNL) function as

$$f_l(i) = 1 + \min(d_x, d_y) + \left\lfloor \frac{d}{2} \right\rfloor [\left\lfloor \frac{d}{2} \right\rfloor + (d\%2) - 1],$$

where $d := d_x + d_y$, $\left\lfloor \frac{d}{2} \right\rfloor$ is the integer quotient, $(d\%2)$ is the remainder of d divided by 2, $d_x := d(i, x), d_y := d(i, y)$ is the distance between nodes i and x,y respectively, prohibiting the usage of the (x, y) edge. With this restriction some nodes i can be unreachable from x or y, resulting in $d(i, x) = \infty$ or $d(i, y) = \infty$. In such a case, they are assigned a "0" label.

After obtaining the label values, they are transformed into a one-hot encoded vector representation, from which we construct the matrix X.

3.4.1.2 Latent features

The authors of [9] say generating the latent features (node embeddings) is non-trivial. They came up with a trick called "negative injection". Presume that we are given the observed network $G = (V, E)$, a set of positive training links $E_p \subseteq E$ and a set of negative links $E_n \cap E = \emptyset$. The reason for this trick is potential data leakage. If we directly generate node embeddings on G, they will record the link existence information of the training links. Authors claim that this would result in bad generalization performance. The trick lies in temporally adding E_n into E and generate the embeddings on $G' = (V, E \cup E_n)$. We talk more about generating E_n in section 4.3.1. The node embeddings are then placed to the corresponding block in matrix X.

3.4.1.3 Explicit Features

There is no distinctive way of treating the explicit node features compared to other data pre-processing for fitting any neural network. These features are placed to the corresponding block in matrix X.

3.4.1.4 Edge features

There is no mention in [9] of how to incorporate edge features into their framework, as the datasets used for experiments in this paper did not have any of this additional information.

However, considering banking data, there is always a possibility of constructing a graph based on transaction interaction between the clients. Edges in this graph would have aggregated features about client-client transaction interaction.

We assume that in the case of predicting client relationships, it makes sense to add these transaction interaction features to the matrix X, and the whole learning framework can stay the same. As the nodes in the enclosed subgraph are structurally labeled, explained in section 3.4.1.1, we can append aggregations – e.g., mean, standard deviation, minimum, maximum – of the transaction interactions and other additional client-client features grouped by these structural labels.

Chapter 4

Analysis and implementation

This chapter will provide an analysis and implementation overview of the primary use case of this thesis – the use of a graph neural network to detect and predict a household relationship between two individuals, defined in 2.1. In the previous chapter 3.4.1, we introduced a theoretical background to the algorithms suited for this task. In this chapter, we would like to present the combined approach that would eventually provide us with an end-to-end solution to this task.

This chapter will follow a Cross Industry Standard Process for Data Mining 4.1 describing all its parts in the context of this thesis.

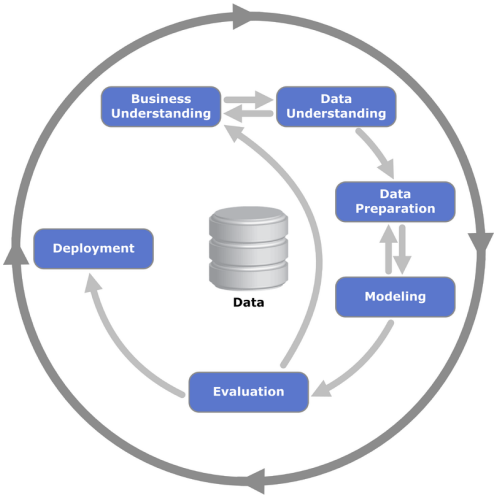


Figure 4.1: A CRISP-DM diagram [1]

4.1 Business understanding

"The Business Understanding phase focuses on understanding the objectives of the project." [16] The users, in this case, are internal bank employees that will consume the information about households. The main objective of this algorithm is to provide a prediction of the relationship existence between two clients.

"The science of Social Network Analysis (SNA) boils down to one central concept – our relationships, taken together, define who we are and how we act." [4] This is why knowing your customer – and his relations to others – is beneficial for both the institution and the clients.

Possessing this information might make a difference in providing a service – e.g., taking up a loan – or its rejection. It would also benefit the customer care processes by maintaining better financial advisory to the clients. Imagine we would detect a new family of three – two parents and a child. We notice that one of the guardians does not have life insurance set up – this means that the financial health of this household might be at risk.

These and other similar uses are in demand, and they would be nearly impossible to handle without a reliable and robust source of knowledge.

As of now, without this extended algorithm, around 68% – see 4.1, of identified households, are consisting only of a single resident. In many cases, it might be right – clients really live in the household alone, or they are the only resident of the household that is a client of our bank (as we are not able to detect clients of other banks). However, the percentage seems suspiciously high. It is expected that the algorithm would help us to merge part of the single-person households into existing ones or merge some of them together.

The fundamental property of this algorithm needs to be a low amount of false positives. In our context, it means a low amount of links should be incorrectly predicted as existing if the ground truth says they are not. This metric is critical to the business, and the adoption of this algorithm relies on this. The users would be afraid to utilize the outputs of this algorithm if there were a high percentage of false positives.

4.2 Data understanding

4.2.1 Data sources

The primary source of data will be clients and their addresses. The success of this algorithm depends on the reliable source of clients' addresses. All places of residency are paired with a unique identifier to provide a comfortable way for data manipulations. Multiple individuals can share the same location identifier but do not have to share the same household. Clients are divided into households – each by sharing the same location key while also having the same surname or sharing a bank product – e.g., checking account or loan.

The first obstacle to tackle is a feminine variation of a surname and its pairing with the masculine version. This can be solved by generating all possible versions of a feminine surname from its masculine version and looking for an intersection of these two sets. Luckily, a reasonable set of rules could be obtained from the institute of Czech language at the Academy of Sciences of the Czech Republic [17]. This algorithm is not in the scope of this thesis, but the author worked on its implementation within the work placement.

Another problem might arise if there are more people with a commonly used surname living at the same address and not sharing a household. Analyzing the data, we decided to ignore this case as there were only a few cases like this.

The output data of these transformations are tuples containing two unique client identifiers, signaling that these two clients share a household.

Another source of data, intended mainly for edge features, we introduced in section 3.4.1.4, is a set of transactions. These vectors are aggregated by the unique client identifiers – each transaction contains two, the sender and receiver – so the edge features are created as:

- # transactions
- # transactions rounded to hundreds
- # transactions with a transactional note
- # transactions for every transaction category
- # transactions for each volume range, defined in section 1.2
- # standing orders
- transaction frequency, defined in section 1.1
- summed volume of transactions
- summed volume of standing orders
- ratio of transacastions and standing orders volume

The output table contains the combination of two unique client ids and specified edge features. Utilizing this table, we can generate simple visualizations of any household, see figure.

4.2.2 Examples and statistics

To obtain the households, we applied the algorithm that utilizes the definition from section 2.1. We were able to find over two million of them, see table 4.1. Most of them – over 68% – were detected as single-person households.

# representatives	# households	% of total
1	2 052 979	68.682%
2	635 905	21.274%
3	200 366	6.703%
4	77 949	2.608%
5	16 529	0.553%
6	4 046	0.135%
7	922	0.031%
8	254	0.009%
9	78	0.003%
10 and more	63	0.002%
Total households	**2 989 091**	
Total clients	**4 354 188**	

Table 4.1: Table of identified households and their size

While in many cases, it might be true that some of the clients really do not share a household with any other person, or the other individuals are not our clients, and we do not possess any information about them, this number is suspiciously high.

On the other hand, large households – above six residents – are represented in small numbers, which is good, as these household sizes are not common in our region. We deep-dived into the outliers, and they were identified mainly as cases of two-generation houses, Vietnamese communities, and nursing homes.

4.3 Data preparation

Often overlooked in favor of the overall outcome, data preparation is the key part of any machine learning pipeline. We learned that it is even more true when dealing with complex graph-like data. We provide an example input data in table 4.2 and figure 4.3 as its graph representation. In table 4.2 we denote

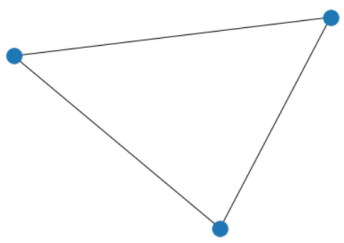


Figure 4.2: Household of size 3

x	y	location_key
1	2	100
2	3	100
4	4	100
5	6	100
5	7	100
6	7	100
8	9	200
8	10	200
9	10	200
11	11	200

Table 4.2: Example table input data

the nodes, between which the edge exist, as x and y. This edge is undirected.

4.3.1 Data preprocessing

In the case of this thesis, the preprocessing takes part in five steps:

- Selecting the data
- Completing the graph to fully connected components
- Generating negative samples
- Checking the graph is made of fully connected components
- Add reversed edges

The first step is to filter data by its relevance to the training. For our case, we only select nodes and all their edges if they are part of a subgraph with two or more nodes. Using the example data, that means nodes 4 and 11 would be excluded entirely from the training dataset.

Completing the graph, so its components are fully connected graphs is not a condition for all graph neural network algorithms. This requirement has arisen from the modeled data itself. As we mentioned in section 2.1 about the household relationship, it would not make much sense if we were to use the raw output data from the household detection algorithm – with missing links – when the nature of this relationship is transitive. Demonstrating on the example data, an edge would be added between nodes 1 and 3.

Although libraries such as PyTorch are ready to help with the complex task of generating the negative samples for graph neural network datasets, for this case, we had to re-implement them ourselves.

This is because the library function would generate a negative link between two random nodes where the link is not observed. Usually, that would be

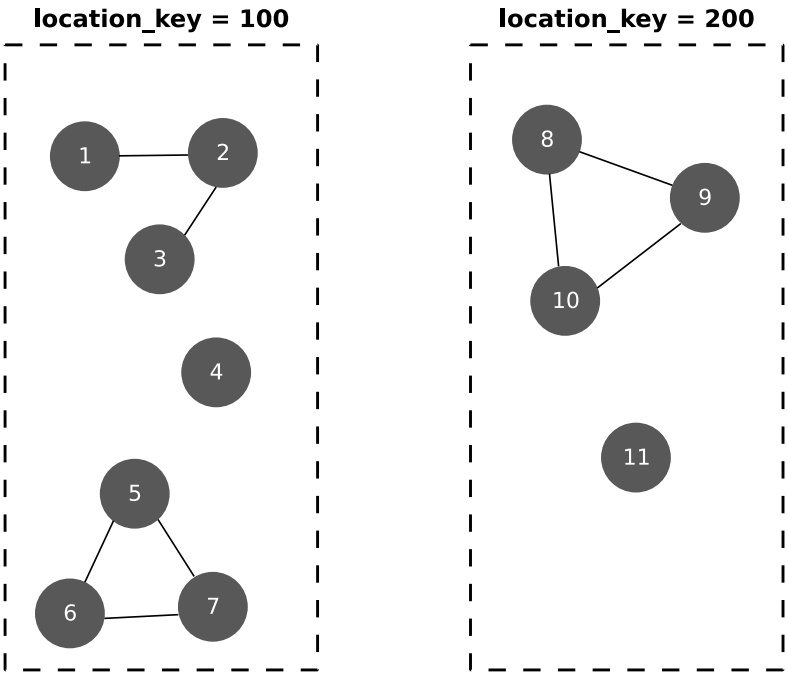


Figure 4.3: Example input data as graphs

x	y	edge_label
1	2	1
1	3	1
2	3	1
5	6	1
5	7	1
6	7	1
1	5	0
1	6	0
1	7	0

x	y	edge_label
2	5	0
2	6	0
2	7	0
3	5	0
3	6	0
3	7	0
8	9	1
8	10	1
9	10	1

Table 4.3: Example output after preprocessing

sufficient and expected. In this use case, we only want to generate negative links between the nodes that share the same location key. This is because of our definition of household from section 2.1. Therefore, after applying all previous preprocessing steps, the negative links – with label 0 – are generated between all nodes with the same location key where the link does not exist.

We provide a code example for steps 1, 2, and 3 4.1. After all these steps, the output would look like a table 4.3. The graph we work with has undirected edges, so the ones with opposite directions are added but not shown to spare space and the reader.

4.3.2 Train, validation, test split

After the preprocessing tasks are done, we can get into a train-test split. As we use the SEAL link prediction framework, described in section 3.4.1, we need to do some further preprocessing as part of the split.

As the first step, all provided edges are assigned to some of the train, validation, and test subsets. In the original source code provided in [18] GitHub repository as an example they are using a Python implementation of:

`torch_geometric.transforms.RandomLinkSplit`

– from the same repository – to handle the job. Part of this class also handles the generation of negative samples.

As we generated our negative samples earlier to suit our case, we already differ in this first step. We had to tweak this class to split positive labeled edges and our negative samples so that the following steps contain the label information. It was not possible with the previous implementation.

After a successful split the algorithm continues for each edge in every data subset in this way:

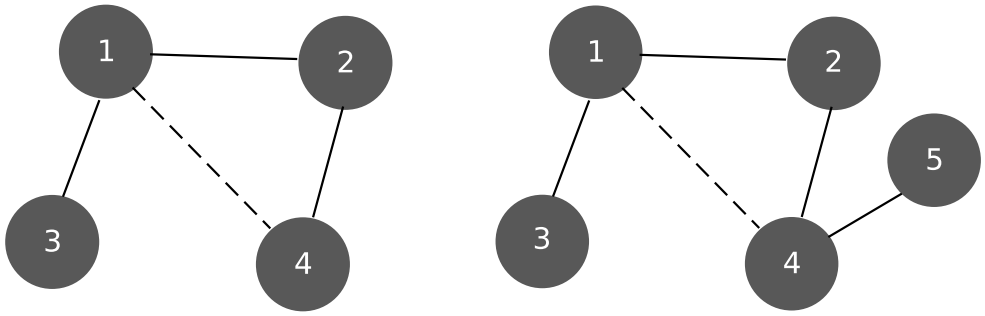
- obtain k-hop subgraph
- remove target link from subgraph
- remove edges so either one of the nodes from target link is of degree 0
- obtain graph embedding
- calculate node labeling
- extract edge features

The k-hop subgraph is a graph that includes the two nodes of the predicted edge and their neighbors as an induced subgraph in the distance of k. This subgraph is obtained by function `torch_geometric.utils.k_hop_subgraph` implemented in [18]. The inputs are the nodes of the target link, the number of hops, and all edges in the data subset. For the number of hops, we have chosen 2. Using just a 1-hop neighborhood would be too trivial, and it would lead to losing some information, as the connected components of which the induced subgraphs are obtained are not fully connected anymore. This function also provides relabeling of the input nodes, so the neural network would not overfit to indices. This means the nodes are given labels from 1 to N, where N is the number of nodes in this subgraph.

The next step is removing the target link from this induced subgraph for the obvious reasons – we want to predict this linkage.

In our use case, we found the next step very important, and it is directly connected to the way how the network will be used. As all edges are divided

into the train, validation, and test subsets, it is not a rule that the k-hop neighborhood is a fully connected component anymore – and it is not because we removed the target link in the previous step. This extraction can lead to the data leakage we had to handle.



(a) Example 1 (b) Example 2

Figure 4.4: Potential data leakage

In figure 4.4 we can see two examples of a k-hop surrounding for predicting the linkage between nodes 1 and 4 – a dashed line means this edge does not exist, but we want to predict it. As the input data might be (almost) fully connected subgraph, we could encounter some data leakage. The transitivity assumption of this relationship still holds, and we had to find a way around it. Modeling these cases and predicting the linkage between nodes 1 and 4 would be, in fact, pointless when there is already a link that connects them to other nodes. We found a way how to solve this problem – remove some edges. We remove links in such a way that the subgraph should consist of two components, where one of the components has only one node of degree 0.

We first look at which one of the nodes from the target link has the lowest degree – to minimize the number of removed edges. If both have the same degree, we choose the node randomly. Then we remove edges that contain this node. However, this might leave us with more than two components in the subgraph. That is why we might add some edges, so the subgraph consists only of two components, each containing one of the nodes that are part of the target link. This removal and addition of edges apply to all possible cases.

In the case of figure 4.4 – example 1 – it would mean that the edge between 2 and 4 would be removed. In the case of example 2, the edge between 2 and 4 could also be removed, the edge between 4 and 5 should then also be removed, and node 5 would be randomly connected with any of the nodes 1, 2, or 3. Example 2 could also be solved by removing edges between nodes 1, 2, and 1, 3 and then randomly connecting node 3 to any of the nodes 2, 4, or 5.

We are using node2vec described and implemented in [12] to obtain the feature representation of the subgraph. The input is a graph representation of the subgraph we generate embeddings for. All edges of this subgraph have the

same weight. This algorithm has two main hyper-parameters – the number of walks per node that we set to 3 and the number of nodes in each walk which we set to 30. Default values are 10 and 80, respectively. We used fewer walks per node with a lower amount of nodes per walk to save computing time for obtaining the corpus. We experimented with various hyperparameter settings, and these were found the best. The implementation from [12] continues with creating a word2vec model implemented in [19]. This model is fed with random walks, and a window – the maximum distance between the current and predicted word within a sentence – is set to 10. The output of this model is a continuous feature embedding of length 64 for each node in the subgraph, see code example 4.2.

In the next stage, we compute node labeling for each subgraph. As mentioned in the section 3.4.1.1, the idea of this label is to signify the role of a node in an enclosing subgraph. In [9] they defined it such as:

1. Center nodes are the target nodes, where we predict the occurrence of a link

2. Nodes with various positions relative to the center have different importance to the link

To do so, we use a function call:

```
drnl_node_labeling(edge_index, src, dst, num_nodes)
```

that is part of repository `example/seal_link_pred.py` in [18]. We only have to provide the edges of the graph, the number of nodes it has, and the source and target nodes that are part of the target link.

The last part of data preparation is extracting the edge features. We could not perform this step earlier as it is dependent mainly on node labeling from the previous step. The original implementations of SEAL, either in [9] or [18] do not implement the use of the edge features, so we incorporate them into this preparation pipeline and add them to the feature vectors of nodes so that the training phase could stay intact.

For each node, the function groups its neighbors – nodes connected to it – by their node labels and performs an aggregation – in our case, their mean value. In this way, the output size of extracted edge features for one node is the input number of features multiplied by distinct values of node labels. If the node does not have neighbors labeled with all of the distinct node label values, an artificial row of zeros is added so that aggregation can be computed because the output for all nodes needs to be of the same length.

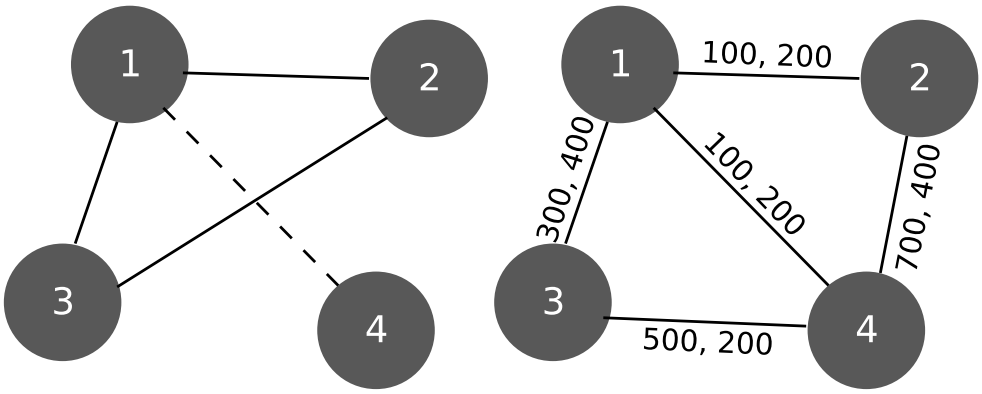
We illustrate these inner workings with an example of the whole process. In figure 4.5 we can see two of the inputs of this function – a graph of the household with link 1-4 we want to predict (left) and a graph of transactional features that we want to extract (right). The graph of transactional features

x	y	feature_1	feature_2
1	2	100	200
1	3	300	400
1	4	100	200
2	4	700	400
3	4	500	200

Table 4.4: Transactional features

is provided as a table 4.4. Other important input to this function is the node labeling from previous step and in this example it can be represented as a dictionary `node_labeling = {1: 1, 2: 0, 3: 0, 4: 1}`. Where keys of the dictionary are nodes and values are their labels. From these inputs the function then produces extracted edge features like in table 4.5. The naming of feature columns denotes the name of feature – `feature_1`, `feature_2` in this case – with additional information, by which label they were grouped. For example, the name `feature_1_0` is the `feature_1` grouped by nodes with label 0.

We then append extracted features to the node features. This section is the most time-consuming part of the whole pipeline. Therefore, after completing this preprocessing, we serialize the datasets into files, so we can skip this part while experimenting with the model. This section concludes the data preparation for modeling.



(a) Household graph (b) Transaction graph

Figure 4.5: Inputs to edge feature extraction

4.4 Modeling

The example repository `example/seal_link_pred.py` in [18] also provides a modeling part that we can apply in our use case.

node	feature_1_0	feature_1_1	feature_2_0	feature_2_1
1	$\frac{100+300}{2} = 200$	$\frac{100}{1} = 100$	$\frac{200+400}{2} = 300$	$\frac{200}{1} = 200$
2	0	$\frac{100+700}{2} = 400$	0	$\frac{200+400}{2} = 300$
3	0	$\frac{300+500}{2} = 400$	0	$\frac{400+200}{2} = 300$
4	$\frac{500+700}{2} = 600$	$\frac{100}{1} = 100$	$\frac{200+400}{2} = 300$	$\frac{200}{1} = 200$

Table 4.5: Extracted edge features

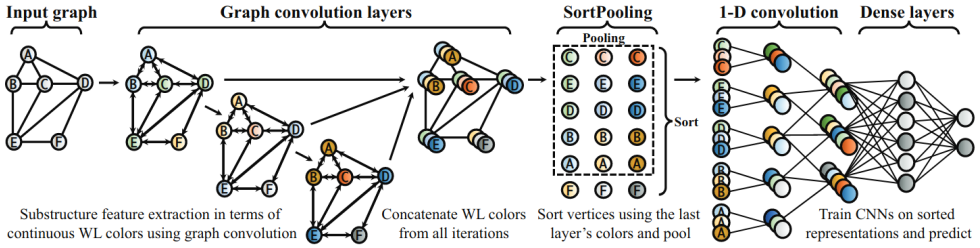


Figure 4.6: The overall structure of DGCNN [2]

It consists of using Deep Graph Convolutional Neural Network (DGCNN) [2] from the same co-author as the SEAL framework [9]. As stated in the [2], their architecture achieved highly competitive performance with state-of-the-art graph kernels and other GNN methods using only graph convolutional layers and their novel SortPooling layer followed by a traditional neural network as the classifier.

We adopted the architecture implemented in `example/seal_link_pred.py` which can be represented by figure 4.6 from the original paper. This architecture was also used in the SEAL framework [9], which we described in section 3.4.1.

This architecture has only two input parameters – the number of hidden channels and the number of layers. For our experiments, we kept the default values of these parameters – 32 hidden channels and 3 layers. First, the input goes through the graph convolutional layers – in our case, 4 of them. The number of these layers is deducted from the input parameter `num_layers`. We explained the main idea behind these in 3.3.1. The algorithm then continues with a novel SortPooling layer. *"The main function of the SortPooling layer is to sort the feature descriptors, each of which represents a vertex, in a consistent order before feeding them into traditional 1-D convolutional and dense layers."* [2] Vertices in the graph are sorted by their structural role within the graph. This ordering is consistent. *"Meaning that vertices in two different graphs will be assigned similar relative positions if they have similar structural roles in their respective graph."*[2]

In [2], they also state another essential property of this layer, besides the

sorting vertex features in a consistent order. The SortPooling layer also unifies the sizes of the output tensors.

These layers are then followed by two traditional 1-D convolutional layers with parameters shown in code example 4.3. They are separated by the MaxPooling layer. We experimented with adding more convolutional layers followed by MaxPooling, but no improvement was discovered. Finally, it is all passed to the Multi-layer Perceptron (MLP) with one hidden layer that performs the binary classification.

In our experiments, we work with only a sub-sample of the available households – about one-twentieth. We found only minor improvements after trying to add more data, so we prefer to save training time and resource consumption.

4.5 Evaluation

In the training of the neural network – and to be able to compare their performance with each other - we use the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) metric. This metric tells us how much the model is capable of distinguishing between our two classes. However, the most important property from the business customer's point of view is a low level of false positives. This is the crucial metric for the algorithm by which it will be decided whether it will be adopted into the deployment phase. The training phase went through 50 epochs, and we trained ten models to report the average performance with a standard deviation.

The ability of the classifier not to label as positive a sample that is, in fact, negative is measured by precision metric. This is calculated as $\frac{tp}{tp+fp}$, where `tp` is the number of true positives and `fp` the number of false positives. We also report the recall score, which is the ability of the classifier to find all the positive samples. This is computed as $\frac{tp}{tp+fn}$, where `tp` is – again – the number of true positives and `fn` the number of false negatives. We also decided to report precision and recall scores for each household size separately. The household size, in this case, means the hypothetical size of the family unit that would occur if the sample was predicted positive. For example, a household of size 4 means that we are trying to predict if a client is a part of a household with only three individuals now.

In the table 4.6 we provide an evaluation of our neural network performance metrics. By looking mainly at precision, we see that the household of size 3 – we predict that some single individual household belongs to a household of size 2 – is performing the best. This household size is the case we expect the most in the evaluation phase, where the ground truth is unavailable. We still provide precision and recall for other sizes of output households. However, we do not expect to extract many relationships between a single individual household and households of size four and more.

	Household of size 3	Household of size 4	Household size 5	Household of size 5+	All households
AUC-ROC	–	–	–	–	0.8477 ± 0.0052
Precision	98.53 ± 0.15	90.14 ± 2.24	76.71 ± 4.21	64.81 ± 6.17	85.94 ± 2.133
Recall	37.82 ± 0.06	49.07 ± 3.73	45.89 ± 1.41	44.79 ± 7.75	43.23 ± 1.8

Table 4.6: Averaged results

If we could obtain data from Czech Statistical Office about the addresses (in our case, locations), we would be able to use them to cross-examine our algorithm performance much better. This data could also be used to clean the training data, which we think would also help.

4.6 Deployment

We think the scores reported in the previous section look promising, and we would like to deploy this model to production. Deployment would enable the customers – consumers of this data – to work with regularly enhanced data and target a bigger audience.

As much as we wanted to make this chapter an end-to-end walk-through of this business use case, we are now still in the phase of deployment. Even if it might seem that after successful development and promising results, the deployment should not take long, the complexity of deployment to the operations of the biggest bank in the country is overwhelming.

However, we do not think that a structural change in a household occurs very often. This model does not need to work in real-time. We think that the outputs could be computed monthly or even quarterly. The outputs must be thoroughly monitored. A particular household should not change in time very often. In the next chapter, we provide a scheme into which the model will be deployed.

```
data = []

# go through subghraphs with the same location key
for name, group in tqdm(df.groupby('loc')):
  G = nx.Graph()
  G.add_edges_from([(r1, r2) for r1, r2
                    in zip(group.x.values, group.y.values)])

  # separate detected households at the location key
  # and filter isolated nodes
  graphs = [G.subgraph(c) for c
            in nx.connected_components(G) if len(c) > 1]

  # add missing edges
  H = nx.Graph()
  for graph in graphs:
    G_nodes = graph.nodes()
    graph_edges_to_add = []
    if len(G_nodes) > 2:
      for i in G_nodes:
        for j in G_nodes:
          if i!=j:
            graph_edges_to_add.append([i,j])

    graph_copy = nx.Graph(graph)
    graph_copy.add_edges_from(graph_edges_to_add)

    # add label of existing edge - 1
    existing_edges = list(nx.to_edgelist(graph_copy))
    H.add_edges_from(existing_edges)
    existing_edges = [[e[0], e[1], 1] for e in existing_edges]
    data.append(existing_edges)

  # add negative samples lebeled as 0
  non_existing_edges = [[e[0], e[1], 0] for e in nx.non_edges(H)]
  data.append(non_existing_edges)
```

Code example 4.1: Preprocessing steps 1, 2 and 3

```
def generate_embeddings(self, nx_G):
  for edge in nx_G.edges():
    nx_G[edge[0]][edge[1]]['weight'] = 1

  G = Graph(nx_G, is_directed=False, p=1, q=1)
  G.preprocess_transition_probs()
  walks = G.simulate_walks(3, 30)
  walks = [list(map(str, walk)) for walk in walks]
  if len(walks) == 0:
    return torch.FloatTensor([0])

  model = Word2Vec(walks, vector_size=64, window=10,
                   min_count=0, sg=1, workers=10, epochs=1)

  my_emb = []
  for node in sorted(list(nx_G.nodes())):
    my_emb.append(model.wv[str(node)])

  my_emb = np.array(my_emb)
  return torch.FloatTensor(my_emb)
```

Code example 4.2: Generating the embedding

```
DGCNN(
  (convs): ModuleList(
    (0): GCNConv(131, 32)
    (1): GCNConv(32, 32)
    (2): GCNConv(32, 32)
    (3): GCNConv(32, 1)
  )
  (conv1): Conv1d(1, 16, kernel_size=(97,), stride=(97,))
  (maxpool1d): MaxPool1d(kernel_size=2, stride=2, padding=0,
                         dilation=1, ceil_mode=False)
  (conv2): Conv1d(16, 32, kernel_size=(5,), stride=(1,))
  (mlp): MLP(32, 128, 1)
)
```

Code example 4.3: DGCNN architecture

Chapter 5

Reconstruction of the network

In this chapter, we would like to introduce the process of reconstruction of the network utilizing the neural network algorithm we produced in chapter 4. In section 4.3 we explained how the training data are prepared and later fed into the neural network. Now we want to use this trained network to predict samples where we do not know the actual label. The preprocessing pipeline for reconstruction is the same as for training data, and we cannot skip any step. One detail is to use already processed mappings for string-like features so that we can represent them as numbers.

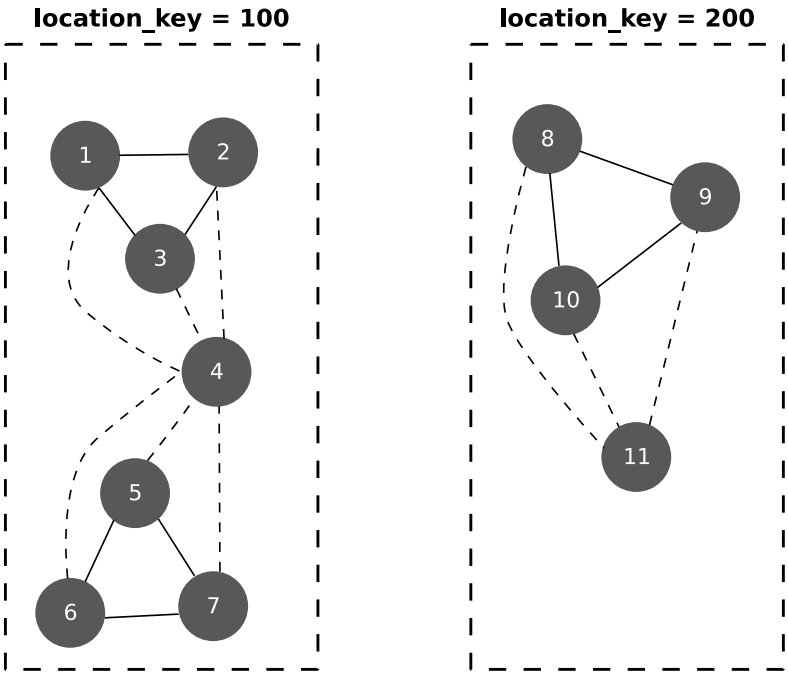


Figure 5.1: Example reconstruction

In figure 5.1 we provide an example reconstruction of the example input from figure 4.3. We try to predict all dashed edges at locations 100 and 200. The order of prediction is the key here. The reconstruction is done locally, meaning we do not try to predict links when the clients do not share the same location key. Then we try to predict the connection of a single household to all other households separately. In the case of `location_key = 100`, we would first predict all dashed links between nodes 1, 2, 3, and 4. If any of these links are predicted as positive, we consider node 4 part of this household. However, links between nodes 5, 6, 7, and 4 must also be evaluated. If any of the predicted edges were positive – we would have a collision. In the cases like this, where a single individual household could be a part of more than one household at the same location, we assign the single household – node 4 – to the household where the predicted linkage was stronger.

The reconstruction is the most time-consuming part of the whole pipeline. The scheme of this process can be summarized as follows:

1. Get all households at location
2. Obtain all households of size > 1
3. Obtain all households of size $= 1$
4. Generate samples to be predicted for a combination of a household of size > 1 and a household of size $= 1$
5. Go through generated samples with all the preprocessing steps
6. Predict all generated samples
7. If any of them is predicted positively – assign a household of size $= 1$ as part of the household of size > 1
8. If there are more combinations of the households of size > 1 and households of size $= 1$, go back to step 4.
9. Solve collisions of households of size $= 1$ being assigned to more than one household of size > 1
10. Continue to the next location

The main bottleneck of this scheme is in step 5 – applying all the preprocessing steps separately for one batch of data. We think it can be solved by enhancing this process with metadata and serialization. It would enable the separation of preprocessing and scoring phases.

With this pipeline, we could broaden the detected households and lower the number of households with a single individual. In table 4.1 we provided the original counts for households respectively to their size. In table 5.1 we

# representatives	# new households
3	119 014
4	14 557
5	1 792
6 and more	255
Total new households	135 618

Table 5.1: Newly identified households

# representatives	# original households	# w projection of new households	% change
1	2 052 979	1 917 361	−7.1%
2	635 905	516 891	−23%
3	200 366	304 823	+52.13%
4	77 949	90 714	+16.38%
5	16 529	18 066	+9.3%
6 and more	5363	5618	+4.75%
Total households	2 989 091	2 853 473	−4.75%

Table 5.2: Identified households before and after applying the algorithm

provide a count of newly identified households and their size. In table 5.2 we project these counts into a total count of all households before and after applying our algorithm.

We see that the total number of households decreased after projecting the newly identified households. That is an expected outcome as we are merging households of size 1 with others of a size greater or equal than 2. The decline is equal to the number of newly identified households. We also see that most newly identified households are of size 3. This sample size was where the model achieved the highest precision in results 4.6.

Chapter 6

Discussion

In this chapter, we want to dedicate a few more sentences to the outcome of this algorithm.

The programming of the neural network started with a baseline. Within the baseline, we used (almost) the same preprocessing and neural network as we used in the final part. We omitted just one step – the edge features 3.4.1.4. Without their existence, the neural network, which works just fine with the academic datasets like Power or USAir [20] did not work with our data – the model was overfitting to the size of the input household. If there were more positive samples with the same household size than negative, the network predicted all links within this household size as positive and vice versa.

We think the results we provide in table 4.6 are reliable, mainly for the assignment to the household of size 2, which is the primary use case as bigger households are not that common. However, the recall score is not that high, and we are still, for sure, missing some linkage.

6.1 Impact

This thesis has a significant impact on our business users. Many households consisting of a single client that we detected with the previous technique – that was helpful for training data – are now assigned to some other households at the exact location. The consumers of this output will now be able to target a broader audience of clients. They have more information about households, which we could not provide before. Now, we can understand the needs of our customers better.

The information will be beneficial for both the clients and the institution. Application ranges from customer care and personalized services to loan applications and their acceptance or rejection.

6.2 Positives

Of course, the most significant positive of this algorithm is that we were able to extract new information – about a part of our clients – that we were missing.

We also consider the fact that we were able to deliver an end-to-end machine learning solution – if we omit the deployment – of a complex problem within the existing infrastructure of the bank without any significant problems as a big plus.

The solution can also serve as a template to work on similar cases in the future, where we can model some other types of relationships from the introductory chapter 2.

6.3 Negatives

We know our solution is not perfect, so we can also pinpoint some of its negatives. The main problem is the computational time spent in the data preprocessing part 4.3, used not only in training but also in evaluation. We consider it a problem mainly in the evaluation part, where we use the scheme mentioned in chapter 5. This scheme could be enhanced by metadata and serialization that would enable separating preprocessing and scoring phases.

The next problem that might occur is the acceptance and deployment of this algorithm. Using neural networks in the banking environment is not considered a standard yet. There is not much we can do other than make sure that the results of our experiments are right and evangelize about the use of neural networks, as complex graph-like data like those we used could be processed very hardly, or even impossibly, with any other machine learning algorithm.

6.4 Limitations and bias

In the data preparation 4.3 section, we only choose subgraphs – households – of size two or more. This filtration is fitted precisely to the use case of predicting if some single household at the exact location is part of another existing household. We simulated this with our preprocessing steps to keep this data out of the training pipeline and only show it to the model in the evaluation phase.

However, the model has not seen a training sample where a linkage between two single-person households would be predicted. It is mainly because we do not have such data where we are hundred percent sure if the link exists or not between these two single-individual households.

This lack of training data does not mean that the model cannot predict such cases, but we currently do not have a way to measure the performance in this case. A possible way would be to sample only those locations where

we detected more than one single-person household, and we have more information about the location – e.g., it is a detached house that is not separated into flats.

Conclusion

The theoretical goal of this thesis was to understand the specifics of transactional banking data and their utilization in the process of social network reconstruction. We could not use this data in their standard tabular form, so we had to find a way of aggregating them into the features that were used in the practical part of this thesis.

After introducing the transactional data, we talked about the ways of linkage prediction and geometric deep learning. We used this knowledge in the practical part, where we introduced an end-to-end framework to predict a linkage of two clients that share a household. To solve this task, we used the SEAL framework [9] which we extended with the ability to use edge features. Adding these features helped us boost the SEAL framework's predictive power and get over 0.84 AUC value with our test out-of-sample dataset.

The last part of this thesis and its practical part is dedicated to evaluating samples we do not have the ground truth labels – network reconstruction. We left households of size 1 out of the training dataset. The idea was then to predict their linkage to already existent households of size greater or equal to 2. We were able to identify over 135 000 of these new households that arose from the merging.

The most common merge we found was between the household of size 1 and the household of size 2, which means we identified a new household of size 3. According to the results on the test dataset, this type of merge performed with over 98% precision score, which is the ability of our classifier not to score a sample that is negative as positive. The low level of false positives is the key criterion for our business partners, who would eventually, after the deployment, consume this data. We think we met this criterion and can proceed to the deployment of this model.

A discussion follows the practical part. We talk about the impact of this model on business and its positives and negatives. We also state the limitations and biases we think we introduced to the model.

Bibliography

[1] Wikipedia, t. f. e. Cross-industry standard process for data mining. 2022, [Online; accessed June 3, 2022]. Available from: https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining#/media/File:CRISP-DM_Process_Diagram.png

[2] Zhang, M.; Cui, Z.; et al. An End-to-End Deep Learning Architecture for Graph Classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018, doi:10.1609/aaai.v32i1.11782. Available from: https://ojs.aaai.org/index.php/AAAI/article/view/11782

[3] Knoke, D.; Yang, S. *Social network analysis*. SAGE publications, 2019.

[4] Kouznetsov, A.; Tsvetovat, M. Social Network Analysis for Startups. 2011, pp. 1–14.

[5] Hamilton, W. L.; Ying, R.; et al. Representation Learning on Graphs: Methods and Applications. *CoRR*, volume abs/1709.05584, 2017, 1709.05584. Available from: http://arxiv.org/abs/1709.05584

[6] Muñoz-Cancino, R.; Bravo, C.; et al. On the combination of graph data for assessing thin-file borrowers' creditworthiness. *CoRR*, volume abs/2111.13666, 2021, 2111.13666. Available from: https://arxiv.org/abs/2111.13666

[7] Sharma, D.; Surolia, A. *Degree Centrality*. New York, NY: Springer New York, 2013, ISBN 978-1-4419-9863-7, pp. 558–558, doi:10.1007/978-1-4419-9863-7_935. Available from: https://doi.org/10.1007/978-1-4419-9863-7_935

[8] Frank, O. Triad count statistics. *Discrete Mathematics*, volume 72, no. 1, 1988: pp. 141–149, ISSN 0012-365X, doi:https://doi.org/10.1016/0012-365X(88)90202-6. Available from: https://www.sciencedirect.com/science/article/pii/0012365X88902026

[9] Zhang, M.; Chen, Y. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, 2018, pp. 5165–5175.

[10] Jeh, G.; Widom, J. SimRank: A Measure of Structural-Context Similarity. KDD '02, New York, NY, USA: Association for Computing Machinery, 2002, ISBN 158113567X, p. 538–543, doi:10.1145/775047.775126. Available from: https://doi.org/10.1145/775047.775126

[11] Arsov, N.; Mirceva, G. Network Embedding: An Overview. *CoRR*, volume abs/1911.11726, 2019, 1911.11726. Available from: http://arxiv.org/abs/1911.11726

[12] Grover, A.; Leskovec, J. node2vec: Scalable Feature Learning for Networks. *CoRR*, volume abs/1607.00653, 2016, 1607.00653. Available from: http://arxiv.org/abs/1607.00653

[13] Bronstein, M. M.; Bruna, J.; et al. Geometric deep learning: going beyond Euclidean data. *CoRR*, volume abs/1611.08097, 2016, 1611.08097. Available from: http://arxiv.org/abs/1611.08097

[14] Sukharev, I.; Shumovskaia, V.; et al. EWS-GCN: Edge Weight-Shared Graph Convolutional Network for Transactional Banking Data. 2020, 2009.14588.

[15] Kipf, T. N.; Welling, M. Variational Graph Auto-Encoders. 2016, 1611.07308.

[16] Alliance, D. S. P. What is CRISP DM? 2022, [Online; accessed June 3, 2022]. Available from: https://www.datascience-pm.com/crisp-dm-2/

[17] Knappová, M. *Přechylování příjmení v češtině (Pravidla a systematický přehled)*. Ústav pro jazyk český AV ČR, v. v. i. Available from: http://nase-rec.ujc.cas.cz/archiv.php?art=6153

[18] Fey, M.; Lenssen, J. E. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. Available from: https://github.com/pyg-team/pytorch_geometric

[19] Rehurek, R.; Sojka, P. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, volume 3, no. 2, 2011.

[20] Rossi, R. A.; Ahmed, N. K. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*, 2015. Available from: https://networkrepository.com

Appendix A

Glossary

POS Point Of Sale

CRISP-DM CRoss-Industry Standard Process for Data Mining

BFS Breadth-First Search

DFS Depth-First Search

GDL Geometric Deep Learning

GCN Graph Convolutional Networks

GAE Graph Autoencoders

SEAL learning from Subgraphs, Embeddings and Attributes for Link prediction

GNN Graph Neural Network

DRNL Double-Radius Node Labeling

SNA Social Network Analysis

DGCNN Deep Graph Convolutional Neural Network

MLP Multi-Layer Perceptron

AUC Area Under Curve

TP True Positive

FP False Positive

Appendix B

Contents of Enclosed CD

- `README.md` ..... the file with CD contents description
- `src`
  - `thesis` ..... thesis source code in LaTeX format
- `text` ..... text práce
  - `thesis.pdf` ..... the thesis ext in PDF