FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Graph-Based Fraud Detection in Recommender Systems
Student:	Bc. Daniel Bohuněk
Supervisor:	Rodrigo Augusto da Silva Alves, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2024/2025

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 11 January 2024 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Instructions

Recommender systems (RSs) is an efficient way of providing personalized experiences for users in web systems. These systems often rely on past interactions between users and items, such as user ratings, item preferences, or user-item reviews. However, RSs are susceptible to attacks from individuals seeking to manipulate recommendations by deliberately generating fake interactions, such as producing fraudulent reviews, to artificially influence the recommender system's behavior(e.g., boosting the popularity of certain items). Sophisticated attacks may be spread over an extended period and across various items, making them challenging to detect.

The primary objective of this study is to investigate the identification of fraudulent reviews or users by utilizing their relationships within graph neural networks.

1. Research anomaly detection methods employed in recommender systems and present a comprehensive literature review in the related work section.
2. Acquaint yourself with graph neural networks for processing user-item interactions.
3. Develop and implement a novel graph-based semi-supervised approach for detecting fraud in recommender systems.
4. Evaluate the performance using at least two diverse datasets and compare the results with a minimum of four existing methods. *
5. Investigate possible interpretable aspects of the developed model.

*For instance, public datasets available in: https://cseweb.ucsd.edu/~jmcauley/datasets.html

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 11 January 2024 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Graph-Based Fraud Detection in Recommender Systems

Bc. Daniel Bohuněk

Department of Applied Mathematics
Supervisor: Rodrigo Augusto da Silva Alves, Ph.D.

May 9, 2024

Acknowledgements

I want to thank my supervisor, Rodrigo Augusto da Silva Alves, Ph.D., for his invaluable guidance and support, as well as my family for their unwavering encouragement throughout my studies.

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 Section 2373(2) of Act No. 89/2012 Coll., the Civil Code, as amended, I hereby grant a non-exclusive authorization (licence) to utilize this thesis, including all computer programs that are part of it or attached to it and all documentation thereof (hereinafter collectively referred to as the "Work"), to any and all persons who wish to use the Work. Such persons are entitled to use the Work in any manner that does not diminish the value of the Work and for any purpose (including use for profit). This authorisation is unlimited in time, territory and quantity.

In Prague on May 9, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Bohuněk, Daniel. *Graph-Based Fraud Detection in Recommender Systems.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstract

Fraudsters attempt to camouflage their behavior to remain undetected. This can make it challenging to design models capable of reliably discovering them, as negative samples may be contaminated with hidden positives. Existing research has shown that taking advantage of relationships between instances using graph convolution improves the detection ability. This work proposes a siamese graph neural network that can be trained in a semi-supervised fashion using a small set of known fraudsters. It shows improved performance over existing methods and increased resilience against camouflaged fraudsters.

Keywords Fraud detection, Recommender systems, Semi-supervised learning, Graph neural networks

Abstrakt

Podvodníci se snaží své chování maskovat, aby zůstali skrytí. To může mít za následek náročný návrh modelů, které je mají spolehlivě detekovat, jelikož část podvodníků může zůstat ukrytá v množině označené za ne-podvodníky. Existující výzkum ukazuje, že grafová konvoluce může přinést vylepšení díky její schopnosti využít vztahy mezi jednotlivými případy. Tato práce navrhuje siamskou grafovou neuronovou síť, kterou lze trénovat semi-supervizovaným učením, kdy je k dispozici jen malá množina známých podvodníků. Tento model projevuje lepší výkon než existující metody a vyšší odolnost proti maskovaným podvodníkům.

Klíčová slova Detekce podvodů, Rekomendační systémy, Semi-supervizované učení, Grafové neuronové sítě

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AE Autoencoder

ALS Alternating Least Squares

AP Averaged Precision

AUC Area Under the Curve

BCE Binary Cross Entropy

BPR Bayesian Personalized Ranking

CHGC Channelwise Heterogeneous Graph Convolution

GCN Graph Convolution Network

GNN Graph Neural Network

HAN Heterogeneous Graph Attention Network

HGT Heterogeneous Graph Transformer

KNN K Nearest Neighbors

LMF Logistic Matrix Factorization

LSTM Long Short-term Memory

MAE Mean Absolute Error

MF Matrix Factorization

MLP Multi-layer Perceptron

MSE Mean Squared Error

RLS Regularized Least Squares

RMSE Root Mean Squared Error

RSS Residual Sum of Squares

SNR Signal-to-Noise Ratio

SVD Singular Value Decomposition

Introduction

The paradox of choice [1] (or choice overload [2]) states that an abundance of options can be counterproductive, requiring more effort and casting doubt on whether the final decision is correct. This phenomenon applies whether ordering food from a menu, picking out clothes to wear, or when shopping for furniture. It can also apply when deciding between different brands of products, choosing which show to watch, what book to read, or what music to listen to. Personalized recommendation helps narrow the selection by learning individual user preferences and suggesting relevant items.

Recommender systems are used by many types of content delivery services, such as social media platforms [3], online movie streaming [4], news sites [5], research papers [6], e-commerce [7], but also restaurants [8], hotels [9], or healthcare [10].

The content recommendation algorithms are often optimized to maximize business-related metrics such as user retention [11]. On some platforms, the content itself is created by other users, who are then compensated for the number of views their content gets (related to the number of ads served). This can lead to the content creators playing a game with the recommendation algorithms to maximize their own profit.

Inevitably, some users will resort to trying to exploit the recommender system for personal gain. For instance, so-called *shilling attacks* [12, 13] are a type of recommender exploitation, or fraud, where the goal is to artificially raise the popularity of certain content. The recommendation algorithm is tricked into suggesting this content to real users, stealing attention from honest content. Other types of recommender system exploits can be in the form of altering keywords (or content tags) to leech onto current trends, stealing the content of others, or recycling old content.

Dishonest content can damage the recommender system's integrity, leading to user dissatisfaction by poisoning their personalized recommendations and ultimately hurting the platform [14, 15]. Due to this, it is in the platforms' best interests to eliminate cases of such fraud, avoid bias, and protect the reliability of a recommender system.

Examples of fraud in recommender systems include individual false reviews [16], bot accounts [17], or deceitful content [18]. False reviews are usually posted to create an impression that the given content is of high quality and can be trusted, which is typical for seller reviews in e-commerce or business

reviews, e.g., restaurants or hotels. Bot accounts can facilitate the spreading of misinformation, such as false reviews, or fabricate engagement, e.g., in comment sections under news articles. Clickbait, fake news, or scam adverts can be considered deceitful content.

Fake (or fraudulent) interactions, users, or content should ideally be an anomaly, which represents only a small percentage of the data by definition. The goal is to identify the anomalies and place less emphasis on that data during suggestion or eliminate those instances entirely if they are deemed too harmful (in case of legal issues such as copyright infringement).

Anomaly detection is often interpreted as an unsupervised learning task, wherein the objective is to assign an outlier score to each instance regarding the distribution of all the samples. Having at least a partial understanding of what makes an instance anomalous can help during the design and implementation of algorithms for their automatic detection.

However, anomaly detection can also be seen as a semi-supervised learning task. In this case, it seeks to automatically learn the difference between anomalous and nominal instances, provided some anomalies have already been uncovered. The difference between supervised and semi-supervised learning is that the semi-supervised methods assume a degree of contamination in the unlabeled instances and use the known anomalies to learn to discriminate the two classes. This is especially true in fraud detection, where fraudsters try to blend in with legitimate instances, and thus, it is challenging to build a quality dataset with balanced classes. It would also be a waste not to utilize the unlabeled instances just because it is too expensive to sort through them.

Most of the existing methods for fraud detection in recommender systems typically focus only on shilling attacks (for example [19, 20, 21, 22]), in which unusual amounts of false reviews are submitted in a short time frame. These types of attacks manifest as an uncommon peak if the number of interactions per item is plotted as a time series. The problem is that these approaches do not cover any other type of fraudulent activity. If an attack occurs over a long period of time, it will not be detected as easily. Additionally, the attacks are even more difficult to discover if the attackers use hijacked accounts instead of mass-creating fresh ones. Unmasking such accounts algorithmically is a challenging task, which is the primary motivation behind using machine learning for this purpose.

Contributions

This research contributes to the field of recommender systems and fraud detection by presenting a novel approach using a siamese graph neural network. This thesis fulfills all of the assignment objectives and lays out areas for future work. The contributions can be outlined as follows:

1. **Literature Review and Background**: The first chapter surveys prior work on fraud detection in recommender systems, emphasizing the necessity of mitigating deceptive behaviors to enhance recommendation accuracy. It describes concepts such as graph neural networks, fraud detection, and relevant evaluation metrics.

2. **Siamese Graph Neural Network Model**: The proposed model, a siamese graph neural network, is introduced in the second chapter. This model is inspired by the success of graph-based architectures in fraud detection and the utility of siamese networks for similarity learning. The siamese network architecture compares pairs of samples, learning latent representations to distinguish known fraudulent behavior from unlabeled instances. Two variants of the model are presented: one employing a static distance metric and another integrating a coupled neural network. Interpretable aspects of the siamese graph neural networks are also discussed.

3. **New Heterogeneous Graph Convolution Operator**: To accommodate irregular graph structures with different node types and relationships, a new heterogeneous graph convolution operator is introduced. This operator is inspired by multi-channel image convolution and explicitly designed to utilize existing homogeneous graph convolution operators.

4. **Experimental Evaluation and Comparative Analysis**: The final chapter includes an ablation study featuring multiple experiments exploring various model parameters and settings, such as graph convolution operators, hidden layer configurations, and loss functions. Performance evaluation of the siamese graph neural network is conducted across four datasets, benchmarking against seven existing graph-based approaches for fraud detection in recommender systems.

Chapter 1

Problem Background & Literature Review

This chapter introduces recommender systems, collaborative filtering, and graph convolution and contains an overview of existing methods for fraud detection in recommender systems. It includes a comprehensive literature review of existing approaches and related methods.

1.1 Recommender Systems

Personalized content recommendations increase user satisfaction, benefiting the content platform by securing retention but also benefiting the users by making the search for relevant content much easier [23]. Typical areas where personalization is used include online streaming services, e-commerce, social media, and targeted advertisement.

Recommender systems need to learn the individual preferences of each user. This can be done by monitoring the users' behavior on the platform [24], having the users fill out short questionnaires [25, 26], or recording the interactions between users and items.

In the case of targeted ads, the ad-serving company can create a shadow profile of the user [27], tracking their behavior around multiple partnered sites. Some of this data can be considered sensitive, raising concerns over security and privacy protection [28]. Surveys show that some users are strictly against the invasion of their privacy, while others tolerate it to be shown relevant, personalized marketing [29, 30, 31].

This section will briefly introduce simple recommender systems operating on one isolated platform, such as an online streaming service where users pay for access to movies. If a user watches a documentary about dolphins, they will be recommended other documentaries about marine life. The idea is that they might not be interested in paying to watch reality shows, and it is in the platform's business interest to keep the user coming back. The same idea applies to e-commerce recommender systems that advertise products based on the user's purchase history.

1.1.1 Definitions

Classic recommender system contains two main entities: a set of users $\mathcal{U}$ and a set of items $\mathcal{I}$ [32]. Users interact only with a handful of items, and the task of the recommender system is to predict which unseen items a user will prefer. Sometimes, the task is to recommend users to other users (social media), items to other items (next-basket), or users to items (advertisements).

Every user $u \in \mathcal{U}$ is associated with their features (a real vector describing their age, gender, etc.) and their past interactions (a vector of size $|\mathcal{I}|$). Similarly, every item $i \in \mathcal{I}$ is associated with its feature vector and interaction vector (of size $|\mathcal{U}|$). There are two distinct types of interactions:

- Implicit (e.g., clicked on, time spent watching/listening/reading),
- Explicit (e.g., like/dislike button, number of stars given).

Figure 1.1 shows an example of explicit interactions between users and items. The rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{|\mathcal{U}| \times |\mathcal{I}|}$ represents the user-item interactions. The goal of the recommender system is to accurately predict the missing values of $\mathbf{R}$.

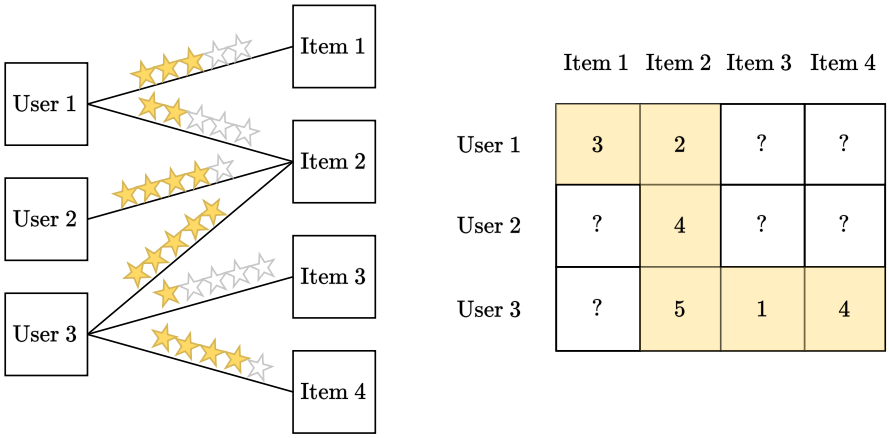
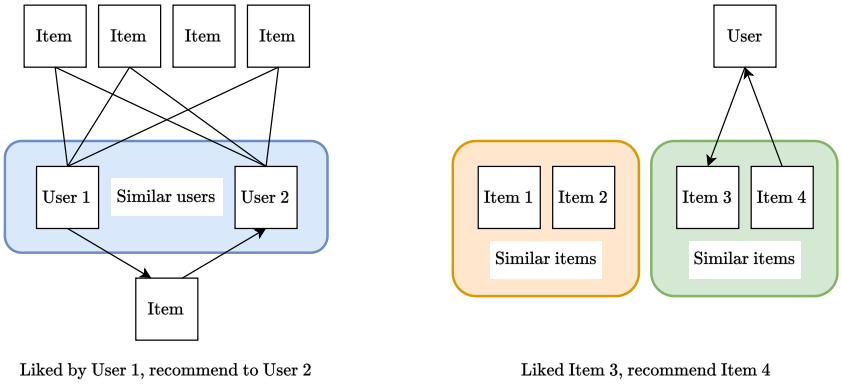


Figure 1.1: Illustration of recommender system interactions and the (sparse) rating matrix. The rows of the rating matrix represent the observed interactions for each user. Likewise, columns represent interactions for items. Explicit 5-star rating interactions.

Content-based filtering uses the item features to suggest new items most similar to those the user positively interacted with previously. Item features (tags, keywords, or embeddings) can be compared using cosine similarity or other distance metrics.

Collaborative filtering relies on other interactions made by other users, utilizing the interaction vectors to find similar users and recommend items that they have liked.

The difference between content-based and collaborative filtering is shown in Figure 1.2. Modern recommender systems typically use a hybrid approach, together with some heuristic strategies such as suggesting currently trending or newly released items.



(a) Collaborative filtering (b) Content-based filtering

Figure 1.2: Illustration of collaborative and content-based filtering recommendation. In collaborative filtering, users can be considered similar if they have similar tastes (or attributes). In content-based filtering, item similarity depends mostly on item attributes.

The interactions between users and items can be natively represented as a bipartite graph, as shown in the left-hand side of Figure 1.1. Recommending unseen items to users translates to predicting missing edges between nodes or predicting missing values in the (sparse) rating matrix $\mathbf{R}$. Rating matrix $\mathbf{R}$ is equal to the biadjacency matrix [33, p. 17] of the bipartite graph.

Apart from the problem of predicting missing ratings in $\mathbf{R}$, other frequently researched problems exist. *Cold-start problem* is when the recommender system must provide recommendations to a fresh user who has yet to interact with any items and their preferences are unknown. Similarly, the same problem can also apply to freshly added items. *Overspecialization*, related to the *filter bubbles* [34], occurs when the recommender system struggles to recommend diverse types of content, which can lead to user alienation if user tastes evolve faster than the recommendations can adapt.

As a reminder, this work is aimed at uncovering fraudulent (or otherwise anomalous) interactions. The motivation behind this is not only to protect legitimate users from fraud but also to prevent bias or distortion of the ratings used for personal recommendations. As [35] shows, penalizing anomalous ratings can improve the recommendation accuracy. The following parts of this section briefly introduce basic recommender system methods, all of which rely on user-item interactions to provide personalized suggestions.

1.1.2 Similarity-based Methods

Similarity can be based on user (or item) attributes or the user-item interactions. Examples of similarity measures that can be used in recommender systems are listed below.

- Cosine similarity for non-zero vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$:

$$\mathcal{S}_{\text{Cosine}}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{||\mathbf{x}|| \cdot ||\mathbf{y}||}.$$

  Equal to the cosine of the angle between the two vectors. Returns values from the interval $[-1, 1]$. $||\cdot||$ operator denotes the ℓ_2 norm, i.e., $||\mathbf{x}|| = \sqrt{\mathbf{x}^T \mathbf{x}}$.

- Euclidean similarity for vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$:

$$\mathcal{S}_{\text{Euclid}}(\mathbf{x}, \mathbf{y}) = \frac{1}{1 + \sqrt{\sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{y}_i)^2}}.$$

  Inverses the Euclidean distance; the closer the two vectors are, the greater the similarity. Returns values from the interval $[0, 1]$.

- Jaccard similarity for non-empty sets $\mathcal{A}, \mathcal{B}$:

$$\mathcal{S}_{\text{Jaccard}}(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|}.$$

  Measures the overlap and intersection of two sets. Returns values from the interval $[0, 1]$.

- Pearson correlation coefficient for non-zero vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$:

$$\mathcal{S}_{\text{Pearson}}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{y}_i - \bar{\mathbf{y}})}{\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})^2 \sum_{i=1}^{n} (\mathbf{y}_i - \bar{\mathbf{y}})}}.$$

  Measures the linear correlation between two vectors. Returns values from the interval $[-1, 1]$. $\bar{\mathbf{x}}$ denotes the mean of $\mathbf{x}$, i.e., $\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$.

Cosine similarity, Euclidean similarity, and Pearson correlation coefficient can measure the similarity of users or items based on past interactions (both implicit or explicit ratings) as well as measure similarity based on the user of item attributes. Jaccard similarity can measure the similarity in item preference based on the sets of items that two users have interacted with but completely ignores any ratings that the users might have given to the items. Cosine similarity deals well with sparse vectors, which are typical in recommendation due to users interacting with only a handful of items in an environment filled with thousands or millions of unique items.

1.1.2.1 K Nearest Neighbors

K Nearest Neighbors (KNN) is generally used for regression or classification tasks but can also be used as a recommender system as described by Algorithm 1.

Additionally, the computed similarities can be used as weights to increase the recommendation accuracy [36]. KNN is relatively straightforward to implement and is easily interpretable. However, it can still suffer due to sparsity and the curse of high dimensionality.

Algorithm 1: KNN for recommendation

```
Function kNNrecommend(Set of users 𝒰, User u, Integer k)
    Compute similarity between user u and all other users from 𝒰
    Select k most similar users
    Aggregate their favorite items as a set ℐ_new
    Recommend items from ℐ_new to user u
end
```

1.1.3 Matrix Factorization

Matrix factorization can be used with the intent to compress data by approximating the original matrix by a lower-rank matrix, acting as dimensionality reduction. A latent representation of user/item ratings can be obtained by applying these methods to the rating matrix $\mathbf{R}$.

In recommender systems, matrix factorization is approached as an optimization problem, where the goal is to minimize the approximation error for the observed values in the sparse rating matrix $\mathbf{R}$ with a dense lower-rank matrix. The values in the dense lower rank matrix can be treated as predicted ratings where initially not observed.

Formally, let $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{m \times n}$ be a sparse matrix with indices of observed values defined by $\Omega = \{(i,j) \mid \mathbf{R}_{i,j} \neq ?\}$. For a given $d \in \mathbb{N}$, find $\hat{\mathbf{R}} = \mathbf{U} \cdot \mathbf{V}$, where $\mathbf{U} \in \mathbb{R}^{m \times d}, \mathbf{V} \in \mathbb{R}^{d \times n}$ such that the loss function $\mathcal{L}(\mathbf{R}, \hat{\mathbf{R}})$ is minimized. In other words, find the solution for:

$$\underset{\mathbf{U},\mathbf{V}}{\operatorname{argmin}} \ \mathcal{L}(\mathbf{R}, \mathbf{U} \cdot \mathbf{V}).$$

Example of loss functions for a sparse matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{m \times n}$ with indices of observed values Ω and matrix $\hat{\mathbf{R}} \in \mathbb{R}^{m \times n}$ are:

- Mean absolute error

$$\mathcal{L}_{\text{MAE}}(\mathbf{R}, \hat{\mathbf{R}}) = \sum_{i,j \in \Omega} |\mathbf{R}_{i,j} - \hat{\mathbf{R}}_{i,j}|.$$

- Mean squared error

$$\mathcal{L}_{\text{MSE}}(\mathbf{R}, \hat{\mathbf{R}}) = \frac{1}{|\Omega|} \sum_{i,j \in \Omega} (\mathbf{R}_{i,j} - \hat{\mathbf{R}}_{i,j})^2.$$

- Root mean squared error

$$\mathcal{L}_{\text{RMSE}}(\mathbf{R}, \hat{\mathbf{R}}) = \sqrt{\frac{1}{|\Omega|} \sum_{i,j \in \Omega} (\mathbf{R}_{i,j} - \hat{\mathbf{R}}_{i,j})^2}.$$

To limit overfitting, choosing $d \ll \text{rank}(\mathbf{R}) \leq \min(m,n)$, and penalizing the values in $\mathbf{U}, \mathbf{V}$ through a regularization parameter $\lambda \in \mathbb{R}^+$ can result in better generalization. Rewritten using MSE, the regularized task is:

$$\underset{\mathbf{U},\mathbf{V}}{\operatorname{argmin}} \ \frac{1}{|\Omega|} \sum_{i,j \in \Omega} \left(\mathbf{R}_{i,j} - \mathbf{U}_{i,:} \cdot \mathbf{V}_{:,j}\right)^2 + \lambda \left(\sum_{i=1}^{m} ||\mathbf{U}_{i,:}|| + \sum_{j=1}^{n} ||\mathbf{V}_{:,j}|| \right).$$

Due to the high sparsity of the rating matrix $\mathbf{R}$, the standard Singular Value Decomposition (SVD) algorithm is not suitable for approximating $\mathbf{R}$ with a lower rank matrix. Instead, iterative algorithms such as Alternating Least Squares (ALS), Bayesian Personalized Ranking (BPR) [37], or Logistic Matrix Factorization (LMF) [38] can be used.

1.1.3.1 Alternating Least Squares

Inspired by the Regularized Least Squares (RLS) solution for ridge regression, ALS solves several linear regression problems per iteration.

As a reminder, the goal of RLS is to find weights $\mathbf{w} \in \mathbb{R}^n$ for $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{y} \in \mathbb{R}^m$ that minimize the regularized Residual Sum of Squares (RSS):

$$\text{RSS}(\mathbf{w}; \lambda) = \sum_{i=1}^{m} (\mathbf{y}_i - \mathbf{X}_{i,:} \cdot \mathbf{w})^2 + \lambda \sum_{i=1}^{n} \mathbf{w}_i^2 = ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 + \lambda \mathbf{w}^T \mathbf{I} \mathbf{w},$$

where $\lambda \in \mathbb{R}^+$ is a regularization parameter and $\mathbf{I} \in \mathbb{R}^{n \times n}$ is the identity matrix.

An explicit solution can be obtained by placing $\nabla_{\mathbf{w}} \text{RSS}(\mathbf{w}; \lambda) = \mathbf{0}$:

$$\begin{aligned}
\nabla_{\mathbf{w}} \text{RSS}(\mathbf{w}; \lambda) &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda \mathbf{I}\mathbf{w} \\
-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda \mathbf{I}\mathbf{w} &= \mathbf{0} \\
2\lambda \mathbf{I}\mathbf{w} &= 2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) \\
\lambda \mathbf{I}\mathbf{w} + \mathbf{X}^T\mathbf{X}\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\
(\lambda \mathbf{I} + \mathbf{X}^T\mathbf{X})\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\
\mathbf{w} &= (\lambda \mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{y})
\end{aligned}$$

ALS matrix factorization algorithm alternates between updating one of the two matrices $\mathbf{U}$ and $\mathbf{V}$, freezing the other one. While updating i-th row of $\mathbf{U}$, the $\mathbf{V}$ is treated as the matrix $\mathbf{X}$, i-th row of $\mathbf{R}$ as the target vector $\mathbf{y}$ and the solution $\mathbf{w}$ is used to update the row $\mathbf{U}_{i,:}$. Equivalently, while updating j-th column of $\mathbf{V}$, $\mathbf{U}$ is treated as the matrix $\mathbf{X}$, j-th column of $\mathbf{R}$ corresponds to the target vector $\mathbf{y}$ and the solution $\mathbf{w}$ is used to update the column $\mathbf{V}_{:,j}$.

Described formally, let $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{m \times n}$ be a sparse matrix with indices of observed values defined by $\Omega = \{(i,j) \mid \mathbf{R}_{i,j} \neq ?\}$. Additionally, let $\Omega^{(k,:)}$ and $\Omega^{(:,l)}$ denote where $\mathbf{R}_{k,:}$ and $\mathbf{R}_{:,l}$ are observed, i.e., $\Omega^{(k,:)} = \{j \mid \mathbf{R}_{k,j} \neq ?\}$ and $\Omega^{(:,l)} = \{i \mid \mathbf{R}_{i,l} \neq ?\}$. For a given hidden dimension $d \in \mathbb{N}$ and regularization parameter $\lambda \in \mathbb{R}^+$, the ALS algorithm initializes $\mathbf{U} \in \mathbb{R}^{m \times d}$ and $\mathbf{V} \in \mathbb{R}^{d \times n}$ randomly and repeats the following steps until convergence (or other stopping criteria, such as a maximum number of iterations):

- Update each row of $\mathbf{U}$:

$$\forall i \in \{1, \ldots, m\} : \mathbf{U}_{i,:} := \underset{\mathbf{u} \in \mathbb{R}^{1 \times d}}{\text{argmin}} \sum_{j \in \Omega^{(i,:)}} (\mathbf{R}_{i,j} - \mathbf{u} \cdot \mathbf{V}_{:,j})^2 + \lambda \sum_{j=1}^{d} \mathbf{u}_{1,j}^2.$$

- Update each column of $\mathbf{V}$:

$$\forall j \in \{1, \ldots, n\} : \mathbf{V}_{:,j} := \underset{\mathbf{v} \in \mathbb{R}^{d}}{\text{argmin}} \sum_{i \in \Omega^{(:,j)}} (\mathbf{R}_{i,j} - \mathbf{U}_{i,:} \cdot \mathbf{v})^2 + \lambda \sum_{i=1}^{d} \mathbf{v}_i^2.$$

The update of one of the rows of $\mathbf{U}$ is illustrated in Figure 1.3. The resulting matrices $\mathbf{U}$ and $\mathbf{V}$ can be used to predict ratings where previously not observed by calculating $\hat{\mathbf{R}}_{i,j} = \mathbf{U}_{i,:} \cdot \mathbf{V}_{:,j}$, getting the predicted rating of i-th user for j-th item. The rows of $\mathbf{U}$ and the columns of $\mathbf{V}$ can be interpreted as latent representations of users and items, respectively.

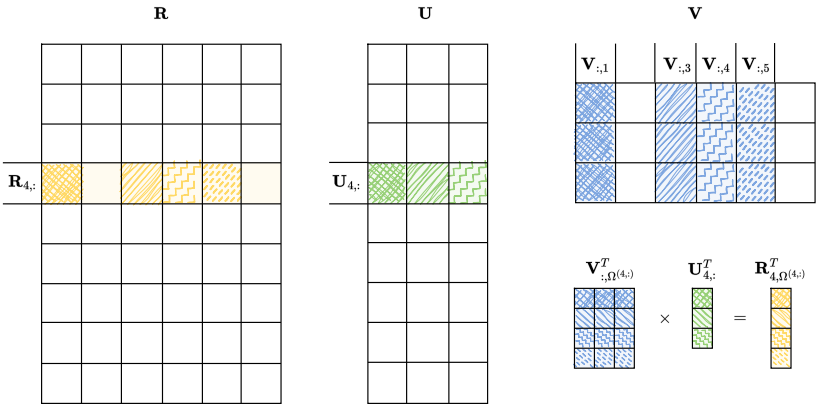


Figure 1.3: Illustration of a row update of the $\mathbf{U}$ matrix in the ALS algorithm. To update the values in row $\mathbf{U}_{4,:}$, the RLS solution can be directly applied as: $\mathbf{U}_{4,:}^T := (\lambda \mathbf{I} + \mathbf{V}_{:,\Omega(4,:)} \mathbf{V}_{:,\Omega(4,:)}^T)^{-1} (\mathbf{V}_{:,\Omega(4,:)} \mathbf{R}_{4,\Omega(4,:)}^T)$.

When choosing the hyperparameters $d \in \mathbb{N}$, $\lambda \in \mathbb{R}^+$, and the stopping criteria (typically maximum number of iterations $k \in \mathbb{N}$), it is possible to split the observed interactions Ω into a train and validation set, Ω_{train} and $\Omega_{\text{validation}}$ to see which combination will provide the best accuracy. Ω_{train} is used to obtain the matrices $\mathbf{U}$ and $\mathbf{V}$, and $\Omega_{\text{validation}}$ is used to measure the error (e.g., MAE, MSE or RMSE, as mentioned previously). To gain an insight into how d and λ influence the reconstruction accuracy, a visualization is provided by Figure 1.4.

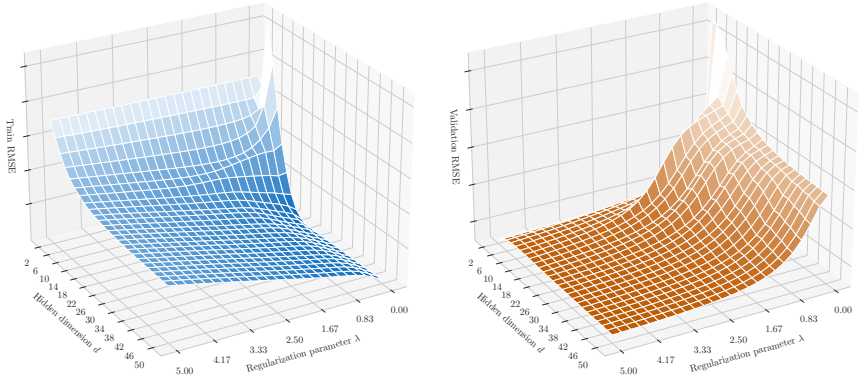
How matrix factorization can be used as a recommender system is described by Algorithm 2.

Algorithm 2: Matrix factorization for recommendation

Function MFrecommend(Rating matrix R, User u)
  Obtain matrix factorization $\mathbf{R} = \mathbf{U} \cdot \mathbf{V}$
  Predict item ratings for u as $\hat{\mathbf{R}}_{u,:} = \mathbf{U}_{u,:} \cdot \mathbf{V}$
  Aggregate items with highest predicted ratings in $\hat{\mathbf{R}}_{u,:}$ as $\mathcal{I}_{\text{new}}$
  Recommend items from $\mathcal{I}_{\text{new}}$ to user u
end

1.1.4 Autoencoders

Much like matrix factorization, Autoencoders seek to learn a latent representation of users (or items). However, instead of creating a temporary vector representation of each user, the autoencoder is trained to recognize the relationships that occur between groups of items through the observed interactions



(a) RMSE on Ω_{train} (b) RMSE on $\Omega_{\text{validation}}$

Figure 1.4: Partial ALS hyperparameter space grid search for $d \in \{2, \ldots, 50\}$, $\lambda \in [0, 5]$, and fixed $k = 25$. As apparent, overfitting occurs when $\lambda \to 0$. Increasing the hidden dimension d has diminishing returns on investment for the validation set. The vertical axes are normalized for both graphs individually for visualization purposes; the actual RMSE values are not the primary focus; their relative scale within a set is.

and is capable of predicting ratings for new users without retraining (unlike matrix factorization, which requires an update every time a new user is added).

Formally, when training an autoencoder, the goal is to minimize the error function $\mathcal{E}$ (reconstruction loss) for a set of vectors $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$:

$$\min_{\theta \in \Theta} \mathcal{E}(\theta) = \min_{\theta \in \Theta} \sum_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\mathbf{x}, \text{AE}(\mathbf{x}; \theta)),$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathcal{L}$ is a loss function (e.g., MSE), AE is the autoencoder function and θ are its parameters from a parameter space Θ.

Autoencoders are typically implemented as deep neural networks, comprised of several hidden layers with a characteristic bottleneck layer in the middle, as illustrated by Figure 1.5. Deep learning utilizes gradient descent to explore the parameter space Θ to arrive at a local minimum.

As a reminder, gradient descent has several versions. However, they all work with a similar premise: calculate the current gradient of the error function $\mathcal{E}$ with respect to the parameters θ, $\nabla_\theta \mathcal{E}$, and use it to update the parameters θ for the next iteration.

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta \mathcal{E}(\theta).$$

- Standard gradient descent utilizes the entire training set $\mathcal{X}$ in each iteration.
- Stochastic (or online) gradient descent estimates the gradient using a single random training sample $\mathbf{x} \in \mathcal{X}$:

$$\nabla_\theta \mathcal{E}(\theta) \approx \nabla_\theta \mathcal{L}(\mathbf{x}, \text{AE}(\mathbf{x}; \theta)).$$

- Minibatch gradient descent estimates the gradient using a random subset $\mathcal{B} \subset \mathcal{X}$:
$$\nabla_\theta \mathcal{E}(\theta) \approx \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \nabla_\theta \mathcal{L}(\mathbf{x}, \mathrm{AE}(\mathbf{x}; \theta)).$$
- Other optimization methods such as Adagrad [39], Adadelta [40], Adam [41], or AdamW [42] are also popularly used. Some gradient descent implementations can also include momentum.

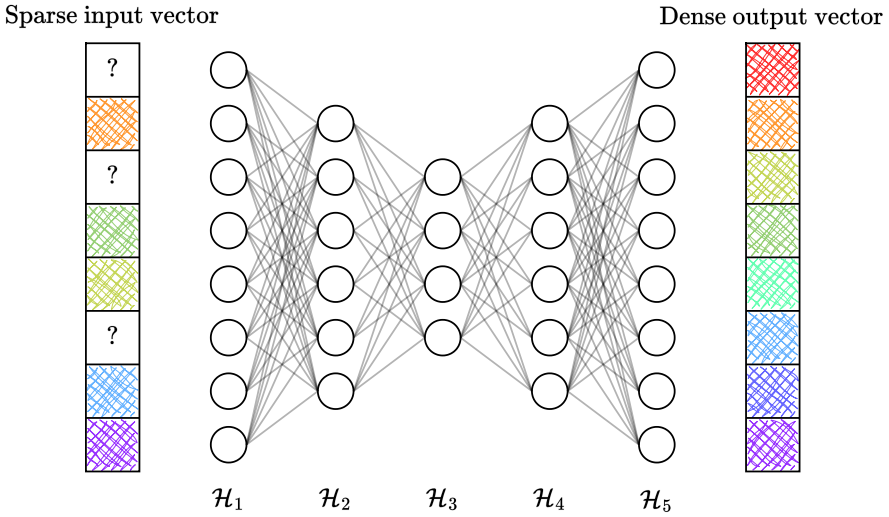


Figure 1.5: Example of an autoencoder neural network used to predict missing values in a sparse vector. $\mathcal{H}_3$ is the bottleneck layer.

For a deep neural network consisting of m hidden layers $\mathcal{H}_1, \ldots, \mathcal{H}_m$, with sizes defined by $\mathbf{n} \in \mathbb{N}^{m+1}$ where each layer $\mathcal{H}_i$ consists of $\mathbf{n}_i$ artificial neurons and an activation function $\varphi^{(i)} : \mathbb{R} \to \mathbb{R}$, the learnable parameters θ include weight matrices $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(m)}$ and bias vectors $\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(m)}$, where $\mathbf{W}^{(i)} \in \mathbb{R}^{\mathbf{n}_i, \mathbf{n}_{i-1}}$ and $\mathbf{b}^{(i)} \in \mathbb{R}^{\mathbf{n}_i}$, the output of i-th hidden layer is computed as:

$$\mathbf{h}^{(i)} = \varphi^{(i)} \left((\mathbf{h}^{(i-1)})^T \mathbf{W}^{(i)} + \mathbf{b}^{(i)} \right),$$

where $\varphi^{(i)}$ is applied element-wise and $\mathbf{h}^{(0)} \in \mathbb{R}^{\mathbf{n}_0}$ is the input vector. Activation functions are most often chosen to be non-linear and continuously differentiable. For non-linear activation functions, it has been proven that multi-layer neural networks are universal approximators [43].

In recommender systems, autoencoders can be trained to reconstruct the interaction vectors of users, i.e., the rows of the rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{m \times n}$. However, the high sparsity can be a problem for loss functions such as MSE. AutoRec [44] proposes a modified loss function, penalizing only observed elements, i.e.;

$$\min_{\theta \in \Theta} \sum_{i=1}^{m} \sum_{j \in \Omega^{(i,:)}} \left(\mathbf{R}_{i,j} - \mathrm{AE}(\mathbf{R}_{i,:}; \theta)_j \right)^2.$$

The authors show that even a shallow autoencoder trained with this loss function can outperform previous matrix factorization methods.

Other significant recommender systems using autoencoders are summarized by a recent survey [45]. How an autoencoder can be used to recommend items to users is described by Algorithm 3.

Algorithm 3: Autoencoder for recommendation

Function AErecommend(Rating matrix R, User u)
  Train an autoencoder AE using rows of matrix $\mathbf{R}$
  Reconstruct the interaction vector $\hat{\mathbf{R}}_{u,:} = \text{AE}(\mathbf{R}_{u,:}; \theta)$ for user u
  Aggregate items $\mathcal{I}_{\text{new}}$ where predicted rating $\hat{\mathbf{R}}_{u,i}$ is high
  Recommend items from $\mathcal{I}_{\text{new}}$ to user u
end

1.1.5 Graph Neural Networks

Graph neural networks (GNNs), first introduced by [46], are able to effectively model the complex relationships that exist between users and items. Unlike the previously mentioned similarity-based methods or matrix factorization, which rely on explicitly defined metrics or matrix decomposition algorithms, GNNs can directly process the structural information contained in the graph representation of user-item interactions. This information is processed via iterative *message passing* between graph vertices (referred to as *graph convolution*), aggregating local neighborhood and global structure context to implicitly create node embeddings, in essence automatically learning additional features for graph entities.

Graph neural networks typically apply a learnable function to the aggregated information for each node, similar to convolutional neural networks that process images. In a way, graph convolutional layers generalize the 2D convolutional layers. Graph convolution operators are explained in greater detail by Section 1.2. GNNs can be used for node, edge, subgraph, graph classification, graph generation, or edge prediction. Edge prediction is most interesting for recommender system applications, as it can serve for predicting missing interactions between user and item nodes in the bipartite user-item graph (illustrated in Figure 1.6).

Existing graph neural network models specialized for recommender systems are outlined by recent surveys [47, 48]. Both conclude that GNN techniques are still maturing and are not without issues. For example, processing dynamically changing graphs and node features requires constant updates as user preferences evolve or new items are added. Large GNNs suffer in scalability, making real-world use, where the number of nodes and edges surpass millions, limited.

Algorithm 4 describes how a link predicting GNN can be used for recommendation. Previously observed interactions can be used as training data for the GNN model.

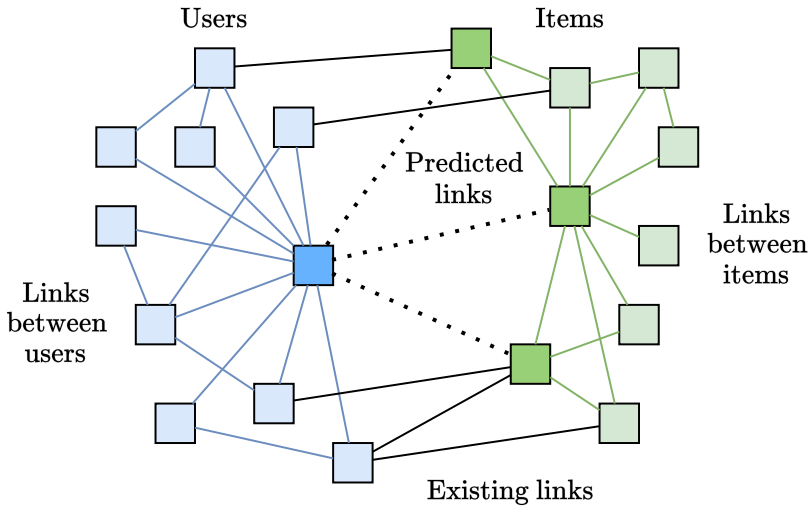


Figure 1.6: Link prediction in user-item graph for recommendation.

Algorithm 4: Link prediction for recommendation

Function GNNrecommend(Set of users $\mathcal{U}$, Set of items $\mathcal{I}$, Interactions $\mathcal{E} = \mathcal{U} \times \mathcal{I}$, User u)
  Collect neighborhood $\mathcal{V}_u \subset \mathcal{U}$ of u
  Construct graph $\mathcal{G} = (\mathcal{V}_u \cup \mathcal{I}, \mathcal{E})$
  Predict most likely links $\mathcal{E}_u = (u \times \mathcal{I})$
  Recommend items from $\mathcal{E}_u$ to user u
end

1.2 Graph Convolution

Image convolution neural networks aggregate information from pixel regions using discrete two-dimensional convolution and learnable filters (sometimes called kernels). This has allowed them to be utilized, for example, in image classification or object segmentation [49]. Let $\mathbf{P} \in \mathbb{R}^{m,n}$ represent a grayscale image and let $\mathbf{K} \in \mathbb{R}^{k,k}$ represent the convolution filter. The convolution operation produces a new image and is denoted as $\mathbf{P}' = \mathbf{P} * \mathbf{K}$, where values of $\mathbf{P}'$ are produced by:

$$\mathbf{P}'_{i,j} = \sum_{a=-\lfloor \frac{k}{2} \rfloor}^{\lfloor \frac{k}{2} \rfloor} \sum_{b=-\lfloor \frac{k}{2} \rfloor}^{\lfloor \frac{k}{2} \rfloor} \mathbf{P}_{i+a,j+b} \cdot \mathbf{K}_{\lfloor \frac{k}{2} \rfloor + a, \lfloor \frac{k}{2} \rfloor + b},$$

where $\mathbf{P}_{i,j} = 0$ for $i \leq 0 \vee i > m \vee j \leq 0 \vee j > n$ (zero-padding, sometimes the resulting image $\mathbf{P}'$ is cropped instead, or other types of padding can be used). This operation is illustrated by Figure 1.7.

Graph convolution, introduced by [50], in a way, generalizes the same concept to graph data. An image can be converted into a graph resembling a grid; each pixel (node) is connected to its neighbors (four neighbors for cen-

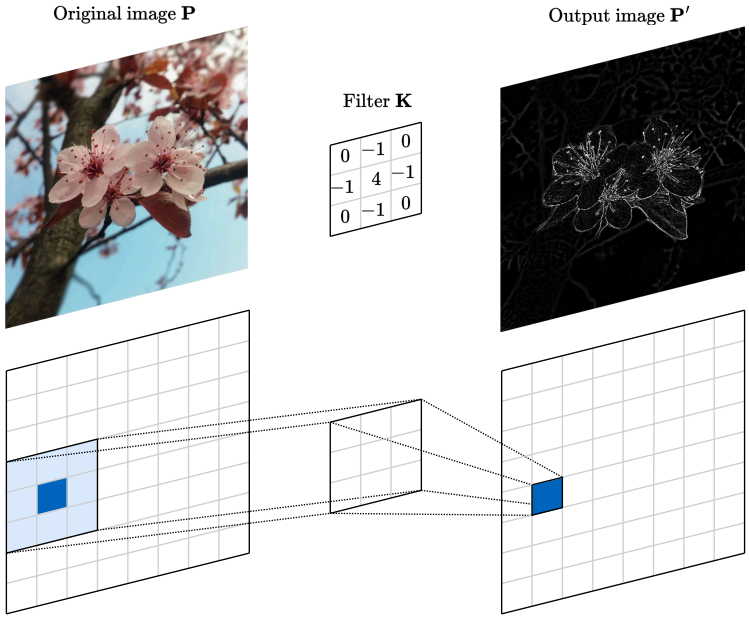


Figure 1.7: Illustration of discrete two-dimensional convolution, including an example of the Laplacean edge-detecting filter. Convolution neural networks learn the weights inside the filter $\mathbf{K}$ during training. The size of the filter, padding strategy, and strides are hyperparameters.

tral pixels, three for border pixels, two for corner pixels). The convolution filter aggregates information from a neighborhood. The same idea is used by graph convolution, wherein information from an irregular node neighborhood is aggregated to produce new node representations.

Graph convolution networks (GCNs), or just graph neural networks (GNNs), use message passing algorithms to propagate information between nodes along graph edges. Message passing takes the following form:

$$\mathbf{p}' = \gamma\left(\mathbf{p}, \bigoplus_{\mathbf{q}\in\mathcal{P}} \xi\left(\mathbf{p}, \mathbf{q}\right)\right),$$

where $\mathbf{p} \in \mathbb{R}^p$ is the original node representation vector, $\mathbf{p}' \in \mathbb{R}^p$ is the updated representation, $\mathcal{P}$ is a set containing neighbors of $\mathbf{p}$, γ and ξ are differentiable functions, e.g., multi-layer perceptrons (MLPs) and $\oplus$ is a permutation invariant aggregation function, e.g., mean, min, max, or sum.

GCNConv operator (introduced by [50]) learns a weight matrix $\mathbf{W} \in \mathbb{R}^{p,q}$ and bias vector $\mathbf{b} \in \mathbb{R}^q$, used by the following algorithm:

$$\mathbf{p}' = \sum_{\mathbf{q}\in\mathcal{P}\cup\{\mathbf{p}\}} \frac{1}{\sqrt{\deg(\mathbf{p})}\cdot\sqrt{\deg(\mathbf{q})}} \cdot \left(\mathbf{W}^T\mathbf{q}\right) + \mathbf{b},$$

where q denotes the size of the output dimension.

GATConv operator (introduced by [51]) uses attention scores $\alpha_{\mathbf{p},\mathbf{q}}$ to determine the importance of each node in a neighborhood. In addition to the weight matrix $\mathbf{W} \in \mathbb{R}^{p,q}$, the attention mechanism contains a weight vector $\mathbf{a} \in \mathbb{R}^{2r}$ and concatenation (denoted by $||$). The operator can be expressed as:

$$\mathbf{p}' = \sum_{\mathbf{q} \in \mathcal{P}} \alpha_{\mathbf{p},\mathbf{q}} \mathbf{W}^T \mathbf{q},$$

where the attention score is computed as softmax of the two concatenated node representations ($\text{LeakyReLU}(\cdot) := \max(\cdot, 0) + \frac{1}{100} \cdot \min(\cdot, 0)$):

$$\alpha_{\mathbf{p},\mathbf{q}} = \frac{\exp\Big(\text{LeakyReLU}\Big(\mathbf{a}^T \cdot [\mathbf{W}^T\mathbf{p} || \mathbf{W}^T\mathbf{q}]\Big)\Big)}{\sum\limits_{\mathbf{r} \in \mathcal{P}} \exp\Big(\text{LeakyReLU}\Big(\mathbf{a}^T \cdot [\mathbf{W}^T\mathbf{p} || \mathbf{W}^T\mathbf{r}]\Big)\Big)}.$$

GATv2Conv operator (introduced by [52]) improves upon this approach by fixing the static attention problem of the original GATConv operator:

$$\alpha_{\mathbf{p},\mathbf{q}} = \frac{\exp\Big(\mathbf{a}^T \cdot \text{LeakyReLU}\Big(\mathbf{W}^T \cdot [\mathbf{p} || \mathbf{q}]\Big)\Big)}{\sum\limits_{\mathbf{r} \in \mathcal{P}} \exp\Big(\mathbf{a}^T \cdot \text{LeakyReLU}\Big(\mathbf{W}^T \cdot [\mathbf{p} || \mathbf{r}]\Big)\Big)}.$$

TransformerConv operator (introducted by [53]) instead uses multi-head dot product attention with weight matrices $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4$ and size of attention heads d:

$$\mathbf{p}' = \mathbf{W}_1^T \mathbf{p} + \sum_{\mathbf{q} \in \mathcal{P}} \alpha_{\mathbf{p},\mathbf{q}} \mathbf{W}_2^T \mathbf{q},$$

where the attention coefficients $a_{\mathbf{p},\mathbf{q}}$ are computed as:

$$\alpha_{\mathbf{p},\mathbf{q}} = \frac{\exp\left(\dfrac{(\mathbf{W}_3^T\mathbf{p})^T(\mathbf{W}_4^T\mathbf{q})}{\sqrt{d}}\right)}{\sum\limits_{\mathbf{r} \in \mathcal{P}} \exp\left(\dfrac{(\mathbf{W}_3^T\mathbf{p})^T(\mathbf{W}_4^T\mathbf{r})}{\sqrt{d}}\right)}.$$

SAGEConv operator (introduced by [54]) uses weight matrices $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$, bias vector $\mathbf{b}$ and a non-linear activation function φ. It aggregates information from neighboring nodes before applying a transformation:

$$\mathbf{p}' = \underbrace{\mathbf{W}_1^T\mathbf{p} + \mathbf{W}_2^T}_{\text{concatenate}} \cdot \underbrace{\left(\bigoplus_{\mathbf{q} \in \mathcal{P}} \varphi\left(\mathbf{W}_3^T\mathbf{q} + \mathbf{b}\right)\right)}_{\text{aggregate}},$$

where $\oplus$ is the aggregation operation (mean aggregation, max pooling, min pooling, or long short-term memory (LSTM) cells).

Other examples of graph convolution operators include ChebConv [55], GraphConv [56], and SSGConv [57]. Additional information on graph neural networks, trends, and practical applications is provided by surveys [58, 59,

60, 61]. Areas where graph neural networks have been successfully used include chemistry, bioinformatics (drug discovery), natural language processing (syntactic and semantic parsing), urban analytics (traffic flow and planning), cybersecurity (network intrusion and fraud detection), knowledge graphs, social networks, and recommender systems.

Graph neural networks can provide advantages by modeling graph-structured data compared to classic tabular data. Much like image convolution neural networks, GNNs have the ability to learn how to capture both global and local patterns, which helps in understanding interactions and relationships between instances (e.g., in social networks) that are lacking in tabular data.

1.3 Fraud Detection

In fraud detection, the goal is to uncover malicious actors, who typically attempt to mask their presence and blend in with other legitimate entities of a system. Otherwise, it is also known under the term *anomaly detection*; however, not all anomalies constitute fraud. Anomaly detection is applicable in tabular data (outlier detection), time series (event detection), images (foreign object detection), or graphs (benign nodes/edges/subgraphs).

Finance is the most notable area where fraud detection is actively combatted due to its direct impact on institutions. Examples of types of fraud include social security fraud [62], credit card fraud [63], insurance fraud [64], loan application fraud [65], audit fraud [66], or (the somewhat new) cryptocurrency fraud [67], showing that as technologies progress, malicious actors evolve together with them, always searching for ways to exploit them for their own gain. That means the fraud detection methods also have to adapt by virtue of the changing environments.

Fraud can also occur in other areas, such as politics (e.g., election fraud [68], corporate lobbying [69], or bribery [70]), telecommunication (e.g., against service provides [71, 72]) and food industry (e.g., ingredients adulteration [73, 74], or their origin [75]), or counterfeit products in general.

Some fraud can be detected by cautious examination by looking for discrepancies. This requires someone who is familiar with the product beforehand. If purchasing goods online, inspecting them in person might not be possible. Instead, the customers have to rely on product reviews (or seller reviews) and base their decisions on them.

Suppose a fraudster is advertising premium brand products at a lower price than other sellers. The fraudster creates false reviews of their products to seem legitimate and lure customers. If the e-commerce platform detects the false reviews, it can punish the fraudster by suspending them. Otherwise, verifying the seller's legitimacy might be challenging before the customers report that they have not received what they had ordered.

Since fraudsters can, over time, adapt to methods that are used to detect them, so too must the detection methods evolve in order to catch them, leading to a neverending cat-and-mouse game. A rule-based approach may be too slow, as the new type of fraud has to get noticed and understood before a rule can be designed to abolish it. On the other hand, machine learning has the potential to model the underlying difference between legitimate instances and fraud, allowing for greater flexibility.

As stated before, fraud detection is related to anomaly detection, as in both cases, the classes are unbalanced. The main difference is that fraudsters are usually agents who make an effort to disguise themselves as legitimate. Anomalous data points stand out from the rest of the data by deviating from what is expected. Anomalies also represent only a small fraction of the dataset, typically less than 5%, as larger portions might not be considered outliers but rather a separate class.

Formally, anomaly detection is a binary classification problem. Let $\mathcal{N}$ be a set of nominal instances and $\mathcal{A}$ be a set of anomalous instances, $|\mathcal{A}| \ll |\mathcal{N}|$. Given the set $\mathcal{X} = \mathcal{N} \cup \mathcal{A}$, anomaly detection methods typically assign each instance $x \in \mathcal{X}$ an outlier score $\eta_x \in \mathbb{R}$ [76], which is used to judge whether x originates from $\mathcal{N}$ or $\mathcal{A}$. Strict decision threshold $\tau \in \mathbb{R}$ can be used:

$$x \in \begin{cases} \mathcal{N} & \text{if } \eta_x < \tau, \\ \mathcal{A} & \text{if } \eta_x \geq \tau. \end{cases}$$

Alternatively, a fuzzy approach can be employed instead, where each instance $x \in \mathcal{X}$ is assigned a probability of it belonging to $\mathcal{A}$:

$$\mathrm{P}(x \in \mathcal{A}) = 1 - \mathrm{P}(x \in \mathcal{N}).$$

Imbalanced classes can present a problem in machine learning, especially when it comes to evaluation. Consider a system that predicts whether a patient is infected with a rare disease occurring in 0.03% of individuals. If the system naively always predicts that a patient is healthy, it will have an accuracy of 99.7%, yet all of the infected individuals will go untreated.

1.3.1 Evaluation Metrics

As previously alluded, it is imperative to consider the context when evaluating the performance of a model in an environment with imbalanced classes. For example, if trying to prevent a deadly disease from spreading, a high false positive rate might be tolerable as long as the false negative rate is minimal. On the flip side, withholding millions of financial transactions in order to prevent a handful of fraudulent ones from going through is also undesirable (sometimes even the delay caused by big data processing is not acceptable, even though it may prevent some fraud cases [77]).

The *confusion matrix* is the most descriptive way of measuring binary classification performance. However, it is not as practical as other derived metrics simply because it consists of multiple values. Confusion matrix is shown in the following diagram:

		Ground truth: Positive	Ground truth: Negative
Prediction	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

Let $\mathbf{y} \in \{0,1\}^m$ denote the ground truth labels (where 0 is negative and 1 is positive) and similarly let $\hat{\mathbf{y}} \in \{0,1\}^m$ denote the predicted labels. Then, the values inside of the confusion matrix can be defined as:

- True positive (TP) $= |\{i \mid \mathbf{y}_i = 1 \wedge \hat{\mathbf{y}}_i = 1\}|$,
- False positive (FP) $= |\{i \mid \mathbf{y}_i = 0 \wedge \hat{\mathbf{y}}_i = 1\}|$,
- False negative (FN) $= |\{i \mid \mathbf{y}_i = 1 \wedge \hat{\mathbf{y}}_i = 0\}|$,
- True negative (TN) $= |\{i \mid \mathbf{y}_i = 0 \wedge \hat{\mathbf{y}}_i = 0\}|$.

These values give the absolute counts for the occurrences of each respective metric. Some variations of the confusion matrix use normalized values (over rows, over columns, or over all values).

The following metrics all utilize the confusion matrix to provide more interpretable values:

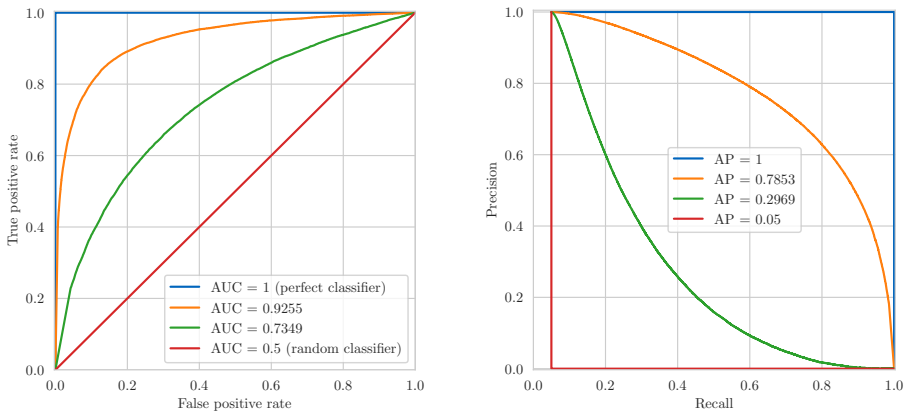
- True positive rate (TPR), also referred to as recall:
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}},$$
- False positive rate (FPR):
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}},$$
- Precision (PRE):
$$\mathrm{PRE} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}},$$
- Accuracy (ACC):
$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}},$$
- F_1 score:
$$F_1 = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})},$$
- G_{mean} score:
$$G_{\mathrm{mean}} = \sqrt{\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} + \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}}}.$$

Value of the decision threshold τ can also be examined (e.g., for models that return the raw outlier score) using the *receiver operating characteristic* (ROC) curve. The ROC curve describes the changing ratio of TPR and FPR for changing τ. An illustration is shown in Figure 1.8a. The ROC curve is directly used by another metric with a self-explanatory name, the *area under the curve* (AUC). AUC can also be interpreted as the probability that the classifier will correctly predict the label of a random sample [78].

Precision-recall curve is similar to the ROC curve, except it measures different metrics. Precision-recall curve is visualized in Figure 1.8b. Averaged precision (AP) utilizes this curve by calculating the weighted mean of the precisions using the difference of the last two recalls as weight:

$$\mathrm{AP} = \sum_{i=1}^{\#\text{ unique } \tau} (\mathrm{TPR}_{\tau_i} - \mathrm{TPR}_{\tau_{i-1}}) \cdot \mathrm{PRE}_{\tau_i}.$$

Some authors argue that the precision-recall curve is better suited for imbalanced datasets than the ROC curve [79], as it better summarizes the prediction accuracy for the anomaly class. In contrast, the ROC curve may show an overoptimistic view.



(a) ROC curve

(b) Precision-recall curve

Figure 1.8: Illustration of ROC and precision-recall curves for an imbalanced dataset, where positive examples constitute 5% of all the data. AUC = 1 and AP = 1 is achieved by a perfect classifier, AUC = 0.5 and AP = 0.05 is achieved by a random classifier.

1.4 Related Works

As described earlier, recommender systems consist of three main entities: users, items, and user-item interactions (implicit or explicit, sometimes including reviews). Each of these may be examined differently, depending on the type of fraud prevention. For example, minimizing the promotion of fraudulent items may differ from catching malicious users who misuse credit card chargebacks or detecting false reviews.

Some platforms deal with the issue of false reviews by allowing the users to rate the reviews themselves. Reviews that obtain too many negative votes are then discarded or hidden, while reviews with positive votes are pushed to the top. However, this approach can still be manipulated the same way (a malicious actor can still promote their own propaganda through other hijacked or bot accounts). Plus, users typically do not care to investigate other activity made by suspicious reviewers (if such information is even available to them) [80]. On the other hand, human moderators employed by the platform could find

such ties. On a platform with thousands or millions of reviews (or comments) submitted daily, the number of moderators would have to be large, and the required coordination might be challenging.

The ultimate goal is to automate the process so that exposure to fraud is minimized for all entities. False reviews should not influence users, the recommender system should accurately suggest legitimate items, and the platform wants to reduce the number of users defrauding it.

1.4.1 Fraud Detection in Recommender Systems

[16] provides a comprehensive review of false review detection methods published between 2007 and 2021. Early works have focused primarily on *opinion fraud* [81, 82], *shilling attacks* [83, 84, 85], or *spam campaigns* [86, 87, 88]. These types of fraud affect both the recommender system users and items. Users could be misled by reading false reviews, and items, in turn, suffer by not being represented accurately and, thus, not being selected by the users. The recommender system itself may be harmed, as the false reviews can cause a bias towards or against certain items, reducing the accuracy of recommendations and ultimately damaging the platform.

[83, 84] rely on time series data modeling to identify unusual events (shilling attacks) that occur in a short time frame. If these fraudulent interactions are spread far apart, this approach might not be suitable. On the other hand, purely analyzing the rating frequencies does not require any feature engineering or breakdown of complex relationships between users and items. Some of the more recent research (for example [89, 90, 91, 92]), which also operate with time series, develop more sophisticated unsupervised methods. Other unsupervised methods use clustering [93, 94] to discern outliers. However, when malicious actors distribute their attacks over a longer period of time or otherwise mask their behavior, these methods fail to recognize them. For this reason, other researchers employ supervised (or semi-supervised) methods that have the potential to learn the intrinsic difference between fraudulent and legitimate instances. Due to the nature of fraud (attackers are attempting to camouflage themselves), accurately labeling enough data for supervised methods is challenging. The method proposed by this thesis uses semi-supervised learning, which can learn the difference between the two classes using only a handful of known fraudulent instances by comparing them against the rest of the unlabeled set.

Other early research utilizes shallow machine learning models, such as logistic regression [81], support vector machines [86, 87], and Naive Bayes [82] to train false review detectors. All of these works use human-labeled datasets and show that their methods are capable of matching human judge performance. Authors of [82] admit that even human judges have trouble deciding whether some reviews are fraudulent or not. These methods rely only on the individual features of the text review and do not take into account other behaviors (e.g., how many reviews have been posted by the same user and what kind of items the user rates positively or negatively). These hidden relationships may have the potential to uncover new patterns that can be used to distinguish fraudsters more easily.

[88, 85] attempt to model various relationships between individual users, items, and groups of users, utilizing those for feature engineering. The authors

propose iterative algorithms which can process the adjacency matrices. Their results show the ability to detect more subtle fraudsters that have escaped previous methods, concluding that the potential that graph features offer can be greatly beneficial in fraud detection.

In recent years, graph neural networks have demonstrated the advantage that learnable node representation brings, using message passing to exchange information via the graph edges. They have proven their ability in recommendation systems (as per surveys [47, 48]), owing to the natural bipartite graph representation of user-item interactions. Similarly, GNNs have been utilized for anomaly detection in graph datasets. [95] provides a comprehensive survey on existing deep learning methods for graph anomaly detection. Most relevant to this thesis are GNN-based recommender system fraud detection methods, such as [96, 97, 98, 99, 100, 101]. Authors of [96] propose a trainable similarity measure to overcome the sparsity of labeled nodes in a semi-supervised setting. [98] expands upon their work, implementing a guided and scalable neighborhood selection and generalizing toward other tasks beyond semi-supervised fraud detection. [99] instead proposes a residual structure to alleviate the dilution of information when training very deep graph neural networks. Authors of [100] demonstrate a novel approach using an additional edge classifier to split the graph into positive and negative subgraphs, later aggregating node representation of both again. However, they focus on spectral analysis, ignoring the issue of camouflaged fraudsters and the need for a large labeled dataset in a fully supervised setting. [97] tries to remedy the class imbalance using a custom balanced graph sampling method but still treats the problem as fully supervised. [101] directly exploits the message passing of GNNs by using labels as a feature but carefully masking them in training to avoid leakage, effectively creating an implicit label propagation algorithm for semi-supervised graph classification. [102] use a dual channel network, in which one module is a traditional multi-layer perceptron processing node features, and the other is a graph neural network processing graph structure. The representations learned by both modules are combined before predicting the node's outlier score.

Table 1.1 contains a summary of the mentioned literature, sorted by the type of approach by their respective authors.

Table 1.1: Structured overview of cited literature on false review detection in recommender systems.

Approach		References
Learning	Strategy	
Unsupervised	Time series	[83, 84, 89, 90, 91, 92]
	Clustering	[93, 94]
Supervised	Shallow models	[81, 82, 86, 87]
	Graph-based	[97, 100, 102]
Semi-supervised		[96, 98, 99, 101]

1.4.2 Siamese Neural Networks

Siamese neural network, introduced by [103], consists of two identical artificial neural networks, conjoining their outputs, which are often compared with each other via some metric or processed by additional neural layers. They have shown great potential in one-shot and few-shot learning scenarios, such as image recognition [104], visual object tracking [105], or user identity tracking [106]. [107] provides a comprehensive survey on the potential of siamese neural networks in recommendation tasks.

Contrastive loss [108] or triplet loss [109] can be used to encourage the model to learn distant embeddings of positive and negative samples. Both losses are defined below (and illustrated by Figure 1.9). Let $d \in \mathbb{R}$ denote the size of embedding vectors and $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is any distance metric.

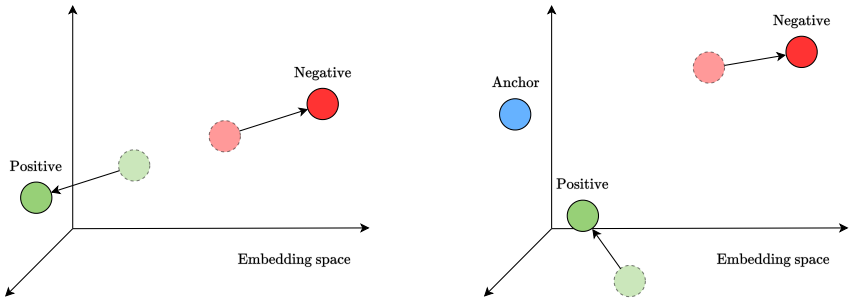
- Contrastive loss, given two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$:

$$\mathcal{L}_{\text{contrastive}}(\mathbf{a}, \mathbf{b}; \alpha) = \begin{cases} \mathcal{D}(\mathbf{a}, \mathbf{b}) & \text{if } \mathbf{a} \text{ and } \mathbf{b} \text{ belong to the same class,} \\ \max(\alpha - \mathcal{D}(\mathbf{a}, \mathbf{b}), 0) & \text{otherwise.} \end{cases}$$

- Triplet loss, given anchor, positive, and negative vectors $\mathbf{a}, \mathbf{p}, \mathbf{n} \in \mathbb{R}^d$:

$$\mathcal{L}_{\text{triplet}}(\mathbf{a}, \mathbf{p}, \mathbf{n}; \alpha) = \max(\mathcal{D}(\mathbf{a}, \mathbf{p}) - \mathcal{D}(\mathbf{a}, \mathbf{n}) + \alpha, 0)$$

For contrastive loss, $\alpha \in \mathbb{R}$ is a hyperparameter defining the lower bound distance between classes. For triplet loss, it defines the minimum offset between different class pairs. Also, it is vital to select difficult negative samples for triplet loss; otherwise, the model will not have much incentive to improve.



(a) Contrastive loss (b) Triplet loss

Figure 1.9: Contrastive and triplet loss illustration. For triplet loss, the objective is to decrease the distance between the anchor and positive samples while increasing the distance between the anchor and negative samples. Contrastive loss only seeks to maximize the distance between positive and negative samples.

Notably, siamese networks have also been utilized for anomaly detection [110], where a fully labeled dataset is challenging to obtain, and few-shot learning can instead take advantage of a smaller set of known anomalies. The authors use triplet loss (illustrated in Figure 1.9) to make their model learn distant representations of positive and negative samples. The distance is subsequently used to determine whether two samples belong to the same class.

[111] proposes learning pairwise relations, i.e., nominal-nominal, nominal-anomaly, and anomaly-anomaly, explicitly designed for semi-supervised anomaly detection environments. The authors effectively augment their training dataset, as they instead predict one of the three classes for a pair of samples, taking significant advantage of a few known positive labels and allowing their model to learn in a supervised end-to-end fashion in a semi-supervised setting. Each pairing is assigned an anomaly score as the target, allowing the optimization to be guided via traditional loss functions, such as MSE or MAE.

Chapter 2

Methodology

This chapter introduces the proposed graph-based semi-supervised method for fraud detection in recommender systems. It describes how heterogeneous graph convolution is implemented, the siamese neural network framework structure, training algorithms, and the necessary steps to correctly process data, including feature engineering.

2.1 Formal Problem Definition

The problem of fraud detection in recommender systems can be formulated as binary node classification. Let $\mathcal{U} = \{\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(m)} \mid \mathbf{u}^{(j)} \in \mathbb{R}^p\}$ be a set of users, $\mathcal{I} = \{\mathbf{i}^{(1)}, \ldots, \mathbf{i}^{(n)} \mid \mathbf{i}^{(j)} \in \mathbb{R}^q\}$ be set of items and $\mathcal{R} = \{\mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(k)} \mid \mathbf{r}^{(j)} \in \mathbb{R}^s\}$ be a set of reviews, meaning each user, item and review is represented by a feature vector and $m, n, k, p, q, s \in \mathbb{N}$, $k \leq m \cdot n$. Every review $\mathbf{r} \in \mathcal{R}$ is associated with a particular user (author) and item, defining a set of relationships $\Psi = \{\psi^{(1)}, \ldots, \psi^{(k)} \mid \psi^{(j)} \in \{1, \ldots, m\} \times \{1, \ldots, k\} \times \{1, \ldots, n\}\}$. Additionally, a tripartite heterogeneous graph $\mathcal{G} = (\{\mathcal{U}, \mathcal{I}, \mathcal{R}\}, \{\mathcal{E}_1, \mathcal{E}_2\})$ can be constructed, with three types of nodes and two types of edges $\mathcal{E}_1 = \{(\psi_1, \psi_2) \mid \forall \psi \in \Psi\}$, $\mathcal{E}_2 = \{(\psi_2, \psi_3) \mid \forall \psi \in \Psi\}$. The graph representation is illustrated by Figure 2.1.

Now, let $\mathbf{y}^{\mathcal{U}} \in \{0, 1\}^m$ denote the ground truth labels for users, where

$$\mathbf{y}_i^{\mathcal{U}} = \begin{cases} 1 & \text{user } \mathbf{u}^{(i)} \in \mathcal{U} \text{ is fraudulent,} \\ 0 & \text{user } \mathbf{u}^{(i)} \in \mathcal{U} \text{ is legitimate.} \end{cases}$$

Similarly, $\mathbf{y}^{\mathcal{I}} \in \{0, 1\}^n$ denote the labels for items, and $\mathbf{y}^{\mathcal{R}} \in \{0, 1\}^k$ for reviews. Given a subset of known fraudulent users $\mathcal{A}^{\mathcal{U}} \subset \{j \mid \mathbf{y}_j^{\mathcal{U}} = 1\}$, the goal is to find a classifier $\mathcal{C}$ with parameters θ, providing the predicted labels $\hat{\mathbf{y}}^{\mathcal{U}} \in \{0, 1\}^n$, in order to maximize an evaluation metric $\mathcal{M}$ (e.g., F_1 score, G_{mean} score, averaged precision, or area under the ROC curve, explained in Section 1.3.1):

$$\underset{\mathcal{C}, \theta}{\operatorname{argmax}} \ \mathcal{M}(\mathbf{y}^{\mathcal{U}}, \mathcal{C}(\mathcal{G}; \theta)).$$

Equivalently, the goal can be to uncover fraudulent items or fraudulent reviews instead, if given some set of previously observed fraudulent items $\mathcal{A}^{\mathcal{I}}$ or fraudulent reviews $\mathcal{A}^{\mathcal{R}}$.

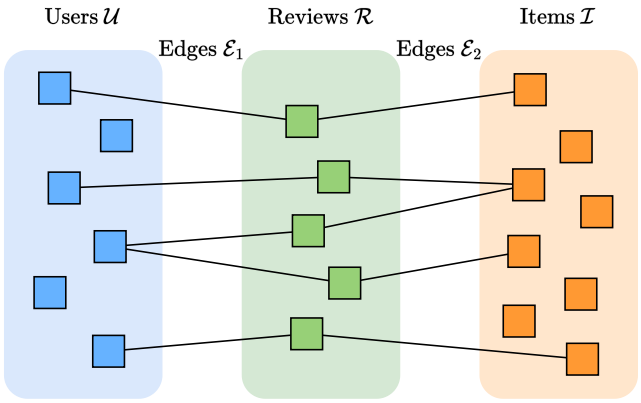


Figure 2.1: Tripartite recommender system heterogeneous graph illustration. Reviews may not be strictly text; other implicit interactions could also be considered as a review (and only represented as a weighted edge). Additionally, there may be hidden relationships between users (e.g., a family) or items (e.g., the same brand), breaking the premise of a tripartite graph.

2.2 Siamese Graph Neural Network

Siamese neural networks process two input samples in parallel, producing two comparable output vectors. Both networks share the same weights, and in practice, a siamese network can be implemented by a single instance, through which two samples are forwarded in series (cutting memory usage in half but doubling the processing time).

The motivation behind using a siamese network is to leverage their few-shot learning capabilities, which have been proven in areas such as facial recognition or object tracking. The model learns by comparing known positive samples against negative ones and maximizing the distance between their embeddings. In a semi-supervised fraud detection setting, only some positive samples are available; the rest are unlabeled. The expected frequency of fraudsters can be used as the probability that a given unlabeled sample is positive or negative. Assuming that fraudulent instances are much less likely than legitimate instances (i.e., it is an anomaly), then if the embedding distance of a sample is very far from all other unlabeled samples, it can be reasonably assumed that it is fraudulent. Coincidentally, it is more likely to be legitimate if it is similar to many other unlabeled samples.

2.2.1 Deep Learning Framework Definition

Given a training graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(m)}\}$, and $\mathcal{A} \subset \{1, \ldots, m\}$ denoting the indices of known fraudulent nodes (the rest of the indices $i \notin \mathcal{A}$ are unlabeled), the goal is to train a function, which assigns a *fraud score* for a pair of node neighborhoods:

$$\phi : \mathcal{G} \times \mathcal{G} \to \mathbb{R},$$

such that $\phi(\mathcal{G}_i, \mathcal{G}_j) > \phi(\mathcal{G}_k, \mathcal{G}_\ell)$ for $(i \in \mathcal{A} \vee j \in \mathcal{A}) \wedge (k, \ell \notin \mathcal{A})$ (where $\mathcal{G}_i$ denotes neighborhood of node $\mathbf{v}^{(i)}$). The function ϕ consists of two modules:

neighborhood embedding function $\phi_1 : \mathcal{G} \to \mathbb{R}^d$ and pairwise fraud scorer function $\phi_2 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. $d \in \mathbb{N}$ is the hidden dimension hyperparameter.

The neighborhood embedding function ϕ_1 is a graph neural network, consisting of graph convolution layers $\mathcal{H}^G$, $|\mathcal{H}^G| = h_G \geq 1$ and optionally also additional linear layers $\mathcal{H}^L$, $|\mathcal{H}^L| = h_L$:

$$\phi_1 = \mathcal{H}^L_{h_L} \circ \cdots \circ \mathcal{H}^L_1 \circ \mathcal{H}^G_{h_G} \circ \cdots \circ \mathcal{H}^G_1.$$

The pairwise fraud scorer can be implemented as a static distance metric (e.g., a Minkowski metric):

$$\phi_2(\mathbf{u}, \mathbf{v}; p) = \sqrt[p]{\sum_{i=1}^{d} |\mathbf{u}_i - \mathbf{v}_i|^p},$$

or for example the signal-to-noise ratio (SNR) based distance metric introduced by [112]:

$$\phi_2(\mathbf{u}, \mathbf{v}) = \frac{\mathrm{Var}(\mathbf{u} - \mathbf{v})}{\mathrm{Var}(\mathbf{u})} = \frac{\sum_{i=1}^{d}((\mathbf{u}_i - \mathbf{v}_i) - \overline{(\mathbf{u} - \mathbf{v})})^2}{\sum_{i=1}^{d}(\mathbf{u}_i - \bar{\mathbf{u}})^2},$$

or similar to what [111] has proposed, a deep neural network with $h \in \mathbb{N}$ layers and parameters $\theta = (\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(h)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(h)}, \varphi^{(1)}, \ldots, \varphi^{(h)})$, where $\mathbf{h}^{(0)} = (\mathbf{u}, \mathbf{v})$ and $\phi_2(\mathbf{u}, \mathbf{v}; \theta) = \mathbf{h}^{(h)}$ and the output of i-th layer is computed as:

$$\mathbf{h}^{(i)} = \varphi^{(i)}\left((\mathbf{h}^{(i-1)})^T \mathbf{W}^{(i)} + \mathbf{b}^{(i)}\right),$$

where $\varphi^{(i)}$ are non-linear activation functions (applied element-wise), $\mathbf{W}^{(i)}$ are weight matrices, $\mathbf{b}^{(i)}$ are bias vectors and ϕ_2 is coupled with the twin graph neural network ϕ_1. Then, using traditional backpropagation algorithms, both ϕ_1 and ϕ_2 can be trained together in an end-to-end fashion.

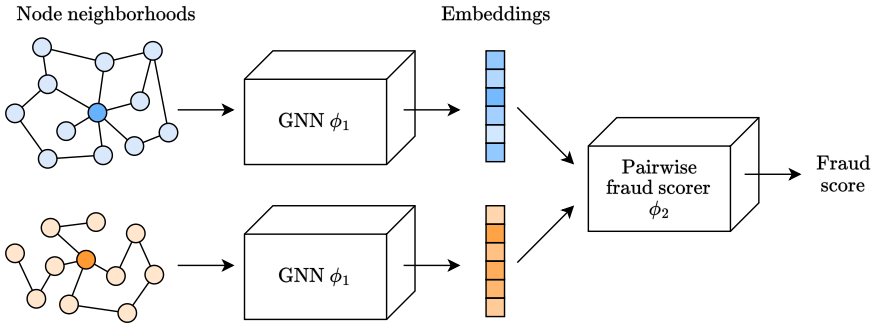


Figure 2.2: Siamese graph neural network architecture. The twin GNNs ϕ_1 share the same parameters, and the pairwise fraud scorer ϕ_2 should be permutation invariant. By comparing a particular sample against many other (unlabeled) samples that are assumed to have low fraud contamination, the model judges which group the sample more likely belongs to by measuring its embedding distance.

2.2.2 Heterogeneous Graph Convolution

Standard message passing algorithms cannot be directly applied on heterogeneous graphs, as the information stored in different node types or relationships expressed by different edge types may not be compatible (they may have unique meanings or even incompatible dimensions). It is possible to discard unwanted node types and treat all edge types as one, sacrificing information. However, such radical transformation may lead to excessive node isolation, especially in bipartite graphs, where each partition consists of different node types, essentially converting graph data into tabular data.

Heterogeneous graph convolution operators already exist, most prominently the Heterogeneous Graph Transformer (HGT) [113], which utilizes attention for determining the contextual importance of nodes in a heterogeneous neighborhood. The authors demonstrate a significant performance advantage over other heterogeneous graph convolution operators, such as Heterogeneous Graph Neural Network (HetGNN) [114] and Heterogeneous graph Attention Network (HAN) [115].

Alternatively, it may be feasible to implement dummy features for each node type and stack multiple graph convolutional layers vertically, one for each type of edge, aggregating or pooling their output to obtain node embeddings (similar to how 2D convolutional layers deal with image color channels). Dummy features pad the original node feature vector with zeroes, such that traditional graph convolution operators may be used, transferring the node's information to neighbors of different types via message passing.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a heterogeneous graph with $\tau_{\mathcal{V}}, \tau_{\mathcal{E}} \in \mathbb{N}$ denoting the number of different types of nodes and edges respectively. $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_{\tau_{\mathcal{V}}}\}$ represents the set of nodes, $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_{\tau_{\mathcal{E}}}\}$ the set of edges, $\mathbf{t} \in \mathbb{N}^{\tau_{\mathcal{V}}}$ describes the size of feature vectors for each node type, i.e., $\mathcal{V}_i \subset \mathbb{R}^{\mathbf{t}_i}$. A particular node $\mathbf{v} \in \mathcal{V}_i$ can be padded with zeroes (dummy features) as:

$$\tilde{\mathbf{v}} := (\mathbf{0}^{(a)}, \mathbf{v}, \mathbf{0}^{(b)}),$$

where

$$\mathbf{0}^{(a)} \in \{0\}^a,\ a = \sum_{j=0}^{i-1} \mathbf{t}_j,\ \ \mathbf{t}_0 = 0$$

$$\mathbf{0}^{(b)} \in \{0\}^b,\ \ b = \sum_{j=i+1}^{\tau_{\mathcal{V}}} \mathbf{t}_j,$$

and additionally, the set of transformed nodes is

$$\tilde{\mathcal{V}} = \bigcup_{i=1}^{\tau_{\mathcal{V}}} \{\tilde{\mathbf{v}} \mid \forall \mathbf{v} \in \mathcal{V}^{(i)}\}.$$

Then, the Channelwise Heterogeneous Graph Convolution (CHGC) can be described as:

$$\mathrm{CHGC}(\mathcal{V}, \mathcal{E}; \varphi, \boldsymbol{\rho}, \oplus) = \varphi\left(\bigoplus_{\tau=1}^{\tau_{\mathcal{E}}} \rho_\tau(\tilde{\mathcal{V}}, \mathcal{E}_\tau; \theta_\tau)\right),$$

where φ is a non-linear activation function, $\oplus$ is an aggregation or a pooling function (e.g., sum, multiplication, mean, min, max, or concatenation), and $\rho_\tau(\tilde{\mathcal{V}}, \mathcal{E}_\tau; \theta_\tau)$ is any homogeneous graph convolution operator with parameters θ_τ (unique for each type of edge $\tau \in \{1, \ldots, \tau_\mathcal{E}\}$), for example any of the operators mentioned in Section 1.2.

CHGC idea is visualized in Figure 2.3. Performance comparison with HGT and HAN in the proposed semi-supervised learning framework will be evaluated later. The heterogeneous graph convolution layers serve to generate node neighborhood embeddings for any type of graph in order to remove unnecessary restrictions on the data graph structure, which could prohibit the modeling of indirect relationships between nodes.

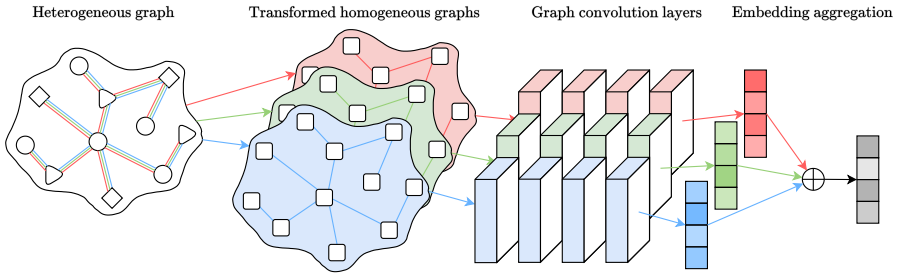


Figure 2.3: Channelwise heterogeneous graph convolution visualization. Separate subgraphs are created for all edge types (channels), and all node types are transformed into one meta-type, creating homogeneous graphs and allowing traditional graph convolution operators to be used.

2.2.3 Model Hyperparameters and Structure

Apart from the usual deep neural network hyperparameters, such as the number of hidden layers and activation functions, there are additional choices for the proposed model: the heterogeneous graph convolution operator used by the twin graph neural network ϕ_1 and the pairwise fraud scorer function ϕ_2. The architecture of the model is shown in Figure 2.2, and all the hyperparameters are listed below:

- Heterogeneous graph convolution ϕ_1
  - CHGC
    - Activation functions (ReLU, LeakyReLU, ...)
    - Homogeneous graph convolution operators (GCNConv, SAGEConv, ...)
    - Aggregation/pooling operator $\oplus$ (sum, mean, max, ...)
  - HGT [113]
    - Number of attention heads
  - Other heterogeneous graph convolution operators (HetGNN [114], HAN [115] ...)
  - Number of graph convolution layers

  - Hidden dimension d (size of neighborhood embedding vector)
- Pairwise fraud scorer function ϕ_2
  - Decoupled static distance metric (Euclidean distance/SNR distance)
  - Coupled learnable deep neural network
    * Number of hidden layers
    * Activation functions

2.2.4 Training

The coupled and decoupled versions cannot be trained in the same way. In the decoupled version, the twin graph neural network ϕ_1 learns to maximize the embedding distance between classes, which is then measured by ϕ_2, while the coupled version learns to output the fraud score directly.

Hierarchical neighborhood sampling from [54] is implemented to fight the scalability issues that plague graph neural networks. This method allows for only a portion of the graph dataset to be loaded, i.e., a neighborhood of a node. During the graph convolution, not all nodes play a part (their messages never reach the central node being considered) and thus can be ignored. As a bonus side-effect, this approach also solves the issue of evolving graphs. If a new node is added (i.e., a new review is posted), its embedding can easily be generated without regenerating embeddings for all nodes in the original graph.

2.2.4.1 Decoupled Version

During training, the decoupled version samples pairs (or triplets) of instances from the training set, forwards their neighborhoods through the twin graph neural network ϕ_1, calculates the contrastive (or triplet) loss using a distance metric $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and adjusts its learnable parameters via backpropagation.

Much like any other model trained using gradient descent, mini-batch training can offer increased efficiency and lead to faster convergence. For this, a special pair (or triplet) sampler must be implemented. The sampling algorithms are described by Algorithm 5. The **SamplePairs** function iterates over all combinations and collects pairs of the same class if their embeddings are further apart than the specified intra-class margin s^+ or if the samples are from different classes and their embeddings are closer than the inter-class margin s^-. These rules are used to prioritize learning on the "difficult" paired instances for faster convergence and to reduce overfitting. The **SampleTriplets** function also iterates over every possible combination. However, it only collects triplets of (anchor, positive, negative), where anchor and positive belong to the same class, while negative belongs to a different class. It also selects only the difficult triplets for which the embedding of the negative sample is closer to the anchor than the embedding of the positive sample (if the minimum embedding distance margin s between anchor-positive and anchor-negative is exceeded). Again, this rule selects triplets that the model currently has trouble with. The mini-batch training algorithm for optimizing parameters θ of the graph neural network ϕ_1 using one of these samplers is described by Algorithm 6.

Algorithm 5: Pair and triplet samplers

Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ // *Number of nodes* $|\mathcal{V}| = m$
Set of indexes of known fraudulent nodes $\mathcal{A} = \{a_1, \ldots, a_p\}$
Set of indexes of unlabeled nodes $\mathcal{N} = \{n_1, \ldots, n_{m-p}\}$
Pre-computed node neighborhood embeddings $\mathbf{X} \in \mathbb{R}^{m,d}$
Function SamplePairs(float s^+, float s^-)
    pairs $\leftarrow \emptyset$
    for $i \leftarrow 1$; $i \leq m$; $i \leftarrow i + 1$ **do**
        for $j \leftarrow 1$; $j \leq m$; $j \leftarrow j + 1$ **do**
            if $((i \in \mathcal{A} \wedge j \in \mathcal{A}) \vee (i \in \mathcal{N} \wedge j \in \mathcal{N})) \wedge (\mathcal{D}(\mathbf{X}_{i,:}, \mathbf{X}_{j,:}) > s^+)$ **then**
                pairs $\leftarrow$ pairs $\cup \{(i, j), (j, i)\}$
            end
            if $((i \in \mathcal{A} \wedge j \in \mathcal{N}) \vee (i \in \mathcal{N} \wedge j \in \mathcal{A})) \wedge (\mathcal{D}(\mathbf{X}_{i,:}, \mathbf{X}_{j,:}) < s^-)$ **then**
                pairs $\leftarrow$ pairs $\cup \{(i, j), (j, i)\}$
            end
        end
    end
    return pairs
end
Function SampleTriplets(float s)
    triplets $\leftarrow \emptyset$
    for $i \leftarrow 1$; $i \leq m$; $i \leftarrow i + 1$ **do**
        for $j \leftarrow 1$; $j \leq m$; $j \leftarrow j + 1$ **do**
            for $k \leftarrow 1$; $k \leq m$; $k \leftarrow k + 1$ **do**
                if $(i, j \in \mathcal{A} \wedge k \in \mathcal{N}) \vee (i, j \in \mathcal{N} \wedge k \in \mathcal{A})$ **then**
                    if $\mathcal{D}(\mathbf{X}_{i,:}, \mathbf{X}_{k,:}) - \mathcal{D}(\mathbf{X}_{i,:}, \mathbf{X}_{j,:}) < s$ **then**
                        triplets $\leftarrow$ triplets $\cup \{(i, j, k)\}$
                    end
                    if $\mathcal{D}(\mathbf{X}_{j,:}, \mathbf{X}_{k,:}) - \mathcal{D}(\mathbf{X}_{j,:}, \mathbf{X}_{i,:}) < s$ **then**
                        triplets $\leftarrow$ triplets $\cup \{(j, i, k)\}$
                    end
                end
            end
        end
    end
    return triplets
end

Algorithm 6: Mini-batch training (decoupled version)

Graph neural network ϕ_1 with learnable parameters θ
Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Learning rate α
Mini-batch size b
Function TrainEpoch(function Sampler)
  Compute current node embeddings $\mathbf{X} \in \mathbb{R}^{m,d}$
  samples $\leftarrow$ **Sampler**()
  samples $\leftarrow$ **shuffle**(**list**(samples))
  for $q \leftarrow 0$; $q < \lceil \frac{m}{b} \rceil$; $q \leftarrow q + 1$ **do**
    // Mini-batch $\mathcal{B}$ of up to b pairs or triplets
    $\mathcal{B} \leftarrow$ samples$[q \cdot b{:}(q+1) \cdot b]$
    if Sampler is **SamplePairs then**
      // Update parameters of ϕ_1 using gradient descent
      $\theta \leftarrow \theta - \alpha \frac{1}{|\mathcal{B}|} \sum_{(i,j) \in \mathcal{B}} \nabla_\theta \mathcal{L}_{\text{contrastive}}(\phi_1(\mathcal{G}_i; \theta), \phi_1(\mathcal{G}_j; \theta))$
    end
    if Sampler is **SampleTriplets then**
      $\theta \leftarrow \theta - \alpha \frac{1}{|\mathcal{B}|} \sum_{(i,j,k) \in \mathcal{B}} \nabla_\theta \mathcal{L}_{\text{triplet}}(\phi_1(\mathcal{G}_i; \theta), \phi_1(\mathcal{G}_j; \theta), \phi_1(\mathcal{G}_k; \theta))$
    end
  end
end

2.2.4.2 Coupled Version

The coupled version uses a regression target for each type of class pairing of node neighborhoods, i.e., unlabeled-unlabeled, unlabeled-fraud, fraud-unlabeled, and fraud-fraud. Given the training graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $|\mathcal{V}| = m$ and set of indexes of known fraudulent nodes $\mathcal{A} = \{a_1, \ldots, a_p\}$ and the set of unlabeled nodes $\mathcal{N} = \{n_1, \ldots, n_{m-p}\}$, the regression target $y \in \mathbb{R}$ for a class pair is defined as:

$$y^{(i,j)} = \begin{cases} 0 & \text{if } i \in \mathcal{N} \wedge j \in \mathcal{N}, \\ 1 & \text{if } i \in \mathcal{N} \wedge j \in \mathcal{A}, \\ 1 & \text{if } i \in \mathcal{A} \wedge j \in \mathcal{N}, \\ 2 & \text{if } i \in \mathcal{A} \wedge j \in \mathcal{A}. \end{cases}$$

Furthermore, the loss function is the mean absolute error (sometimes referred to as ℓ_1 loss):

$$\mathcal{L}_{\text{MAE}}(\phi(\mathcal{G}_i, \mathcal{G}_j; \theta), y^{(i,j)}) = |\phi(\mathcal{G}_i, \mathcal{G}_j; \theta) - y^{(i,j)}|.$$

The mini-batch training differs from the decoupled version in that pairs in every mini-batch are selected randomly. The regression target corresponding to a pair of input samples effectively augments the training data size from m to m^2. Algorithm 7 describes the pair sampling and epoch training.

Algorithm 7: Mini-batch training (coupled version)

Siamese graph neural network ϕ with learnable parameters θ
Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ // *Number of nodes* $|\mathcal{V}| = m$
Set of indexes of known fraudulent nodes $\mathcal{A} = \{a_1, \ldots, a_p\}$
Set of indexes of unlabeled nodes $\mathcal{N} = \{n_1, \ldots, n_{m-p}\}$
Learning rate α
Mini-batch size b
Number of mini-batches per epoch c
Function SampleMiniBatch()
    pairs $\leftarrow \emptyset$
    for $k \leftarrow 0$; $k < \lfloor \frac{b}{2} \rfloor$; $k \leftarrow k + 1$ **do**
        // $i, j \in \mathcal{N}$ randomly chosen
        pairs $\leftarrow$ pairs $\cup \{(i, j, 0)\}$
    end
    for $k \leftarrow 0$; $k < \lceil \frac{b}{8} \rceil$; $k \leftarrow k + 1$ **do**
        // $i \in \mathcal{N}, j \in \mathcal{A}$ randomly chosen
        pairs $\leftarrow$ pairs $\cup \{(i, j, 1)\}$
    end
    for $k \leftarrow 0$; $k < \lceil \frac{b}{8} \rceil$; $k \leftarrow k + 1$ **do**
        // $i \in \mathcal{A}, j \in \mathcal{N}$ randomly chosen
        pairs $\leftarrow$ pairs $\cup \{(i, j, 1)\}$
    end
    for $k \leftarrow 0$; $k < \lceil \frac{b}{4} \rceil$; $k \leftarrow k + 1$ **do**
        // $i, j \in \mathcal{A}$ randomly chosen
        pairs $\leftarrow$ pairs $\cup \{(i, j, 2)\}$
    end
    return pairs
end
Function TrainEpoch()
    for $k \leftarrow 0$; $k < c$; $k \leftarrow k + 1$ **do**
        $\mathcal{B} \leftarrow$ **SampleMiniBatch()**
        $\theta \leftarrow \theta - \alpha \frac{1}{|\mathcal{B}|} \sum_{(i,j,y) \in \mathcal{B}} \nabla_\theta \mathcal{L}_{\mathrm{MAE}}(\phi(\mathcal{G}_i, \mathcal{G}_j; \theta), y)$
    end
end

2.2.5 Inference and Interpretability

The proposed siamese graph neural network compares two samples against each other to determine the fraud score. Due to the inherent contamination of the unlabeled portion of the training set (and unknown labels of new samples), there is a chance for a false positive reading. To obtain a reliable fraud score, a particular sample must be compared against multiple others.

The binary fraud classifier can be defined using hypothesis testing. Given the siamese graph neural network $\phi : \mathcal{G} \times \mathcal{G} \leftarrow \mathbb{R}$ with parameters θ (where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a heterogeneous graph), trained using a set of known fraudulent nodes $\mathcal{A} = \{a_1, \ldots, a_p\}$ and a set of unlabeled nodes $\mathcal{N} = \{n_1, \ldots, n_{m-p}\}$, the mean fraud score of node $u_i \in \mathcal{N}$ is calculated as:

$$\phi(\cdot, \mathcal{G}_{u_i}; \theta) = \frac{1}{m-p-1} \sum_{u_j \in \mathcal{N}, u_j \neq u_i} \phi(\mathcal{G}_{u_j}, \mathcal{G}_{u_i}; \theta).$$

Under the assumption that these scores are sampled from a Gaussian distribution, i.e., $\phi(\cdot, \mathcal{G}_{u_i}; \theta) \sim \mathrm{N}(\mu_{\mathcal{N}}, \sigma^2_{\mathcal{N}})$, the two-sample t-test can be used. The two parameters $\mu_{\mathcal{N}}$ and $\sigma^2_{\mathcal{N}}$ can be approximated using the sample mean and covariance:

$$\mu_{\mathcal{N}} \approx \frac{1}{m-p} \sum_{i=1}^{m-p} \phi(\cdot, \mathcal{G}_{u_i}; \theta),$$

$$\sigma^2_{\mathcal{N}} \approx \frac{1}{m-p-1} \sum_{i=1}^{m-p} \left(\phi(\cdot, \mathcal{G}_{u_i}; \theta) - \bar{\mu}_{\mathcal{N}}\right)^2.$$

Since the fraud contamination should be low ($< 5\%$ ideally), the prevalent instances in $\mathcal{N}$ belong to the legitimate class. With this in mind, obtaining k pairwise fraud scores for some new sample v: $\mathbf{v} = (\phi(\mathcal{G}_u, \mathcal{G}_v; \theta) \mid u \in \mathcal{N})$, a one sample t-test can be performed on whether $\mu_{\mathcal{N}} \geq \bar{\mathbf{v}}$ (null hypothesis) or $\mu_{\mathcal{N}} < \bar{\mathbf{v}}$ (alternative hypothesis). The test statistic

$$T_v = \frac{\bar{\mathbf{v}} - \mu_{\mathcal{N}}}{\sqrt{s^2_k}} \sqrt{k},$$

where $\bar{\mathbf{v}} = \frac{1}{k} \sum_{i=1}^{k} \mathbf{v}_i$ and $s^2_k = \frac{1}{k-1} \sum_{i=1}^{k} (\mathbf{v}_i - \bar{\mathbf{v}})^2$ is used to evaluate the test. The null hypothesis is rejected if $T_v \leq -t_{\alpha,k-1}$, where $t_{\alpha,k-1}$ is the critical value of Student's t-distribution with $k-1$ degrees of freedom, which can be interpreted as meaning that the fraud scores of sample v are significantly higher than other unlabeled samples. Finally, the binary classifier $\mathcal{C}$ can be described as:

$$\mathcal{C}(\mathcal{G}_v; \theta, \alpha, k) = \begin{cases} 1 & \text{if } T_v \leq -t_{\alpha,k-1}, \\ 0 & \text{otherwise,} \end{cases}$$

where k should be sufficiently large ($k \geq 20$) as to reduce the influence of the camouflaged fraudsters within $\mathcal{N}$ and α (significance level) dictates the maximum false positive rate, typically $\alpha = 0.05$. Additionally, the p-value $\inf\{\alpha \mid T_v \leq -t_{\alpha,k-1}\}$ can be interpreted as the inverse confidence in the predicted result (i.e., a p-value of 0.03 means 97% confidence of the classifier that v is fraudulent).

However, as Figure 3.5 shows, the mean fraud scores of unlabeled nodes $u_i \in \mathcal{N}$ more closely resemble an exponential distribution rather than a Gaussian distribution, calling into question whether the two-sample t-test is appropriate. Instead, a non-parametric statistical test can be used, such as the Mann-Whitney U-test [116] or the two-sample Kolmogorov-Smirnov test [117].

Alternatively, the pure sample mean fraud score $\bar{\mathbf{v}}$ can be used as the *outlier score*, delegating the binary classification upon a set threshold. The entire prediction pipeline is visualized in Figure 2.4. The decision threshold can be set arbitrarily, e.g., to maximize a selected metric (on the training or the validation set). While this approach does not offer the same confidence measure as the previous one, the individual contribution of paired samples can be examined instead. Suppose some subset of the k randomly selected unlabeled samples heavily influence the arithmetic mean (towards one side or the other). In that case, they offer some explainability for the outlier score - that v must be very similar (or dissimilar) to this subset, and whether it is legitimate or fraudulent can be decided (depending on the result).

GraphLIME [118] can be used to provide additional interpretability for a graph neural network's output. It can explain the most representative features that influence the final prediction or embedding for each node.

2.3 Data Preparation

A heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ may be constructed using all three types of nodes ($\mathcal{V} = \{\mathcal{U}, \mathcal{I}, \mathcal{R}\}$ for users $\mathcal{U}$, items $\mathcal{I}$, and reviews $\mathcal{R}$) and at least two types of edges $\mathcal{E} = \{\mathcal{E}_{\mathcal{U}\times\mathcal{R}}, \mathcal{E}_{\mathcal{R}\times\mathcal{I}}\}$. Additional edge types can be defined where the relationships make sense, for example:

- Users listed on the same address (or city).
- Users registered around the same time period.
- Items of the same brand (or manufacturer).
- Items based on category and price.
- Reviews posted on the same date.
- Reviews with the same explicit rating (e.g., number of stars).

Suppose some dataset contains no user or item information beyond their identifiers. In that case, nodes $\mathcal{V}$ are all of the same type, and the additional relationship edges are necessary in order to construct a graph and take advantage of message passing algorithms.

During training, validation, and testing, it is imperative to correctly mask data to prevent information leakage between the different sets. This is especially true in graph neural networks, where careful consideration must be taken to the nodes and edges that the model is exposed to if working with a single global graph, as the message passing algorithms can leak information from one set to another. An effective way to prevent this is to create subgraphs for each set, where the training graph contains only nodes from the training set, the validation graph contains nodes from training and validation sets, and the testing graph contains all nodes.

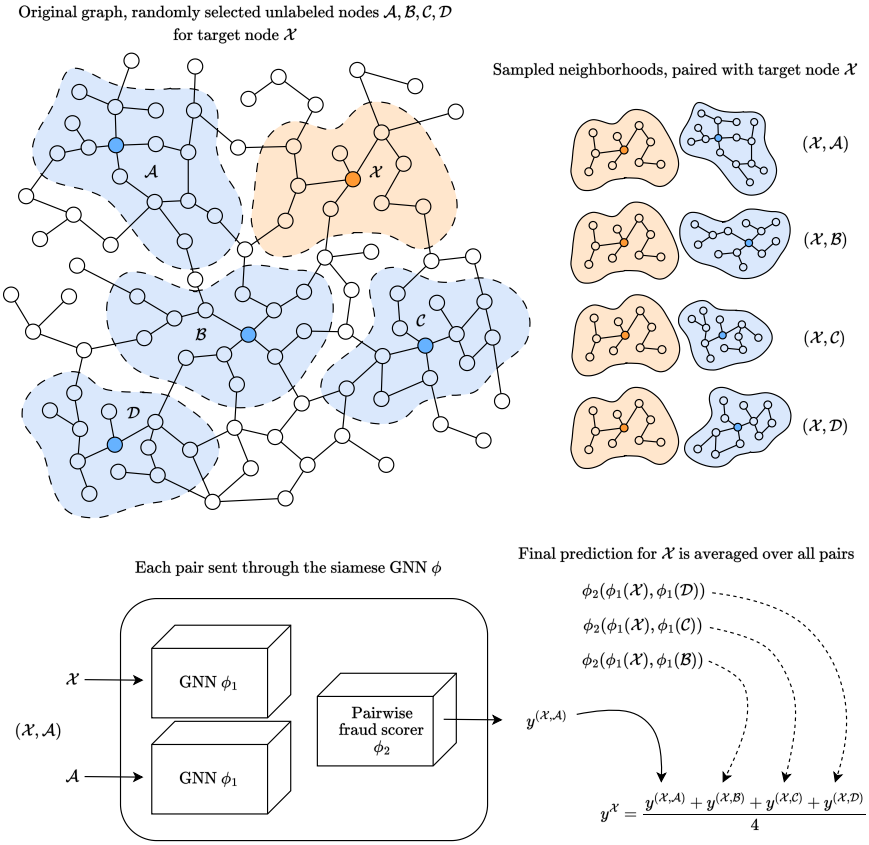


Figure 2.4: Siamese graph neural network prediction pipeline, divided into four steps. Given the target node $\mathcal{X}$, k (in this case $k = 4$) nodes from the unlabeled set are selected. Then, their neighborhoods are sampled and paired with the neighborhood of $\mathcal{X}$. Each of these pairs is forwarded through the siamese GNN, giving k output values, which are averaged to obtain the final fraud score.

2.3.1 Feature Engineering

Depending on the available information, different kinds of features can be produced for each type of entity in the recommender system. Suitable user features may be account creation date, number of reviews, ratio of positive/negative reviews, average rating, rating variation, rating entropy, and average time between reviews. For the reviews, generating features depends on how detailed they are. If reviews are typical text comments, natural language processing techniques, e.g., sentiment analysis or word embedding, might be required. Otherwise, only the explicit (or implicit) rating value and meta information, such as the date and time when the interaction occurred or the device/browser user agent. Item features generally depend on the platform domain; for example, an online streaming service offers movies, while an e-commerce platform offers physical products; movies have actors, directors, and genre; products are

manufactured by a particular brand, have a color, material composition, size dimensions, and possibly a taste or a smell.

Latent representations of users (or items) generated by autoencoders or matrix factorization methods can also be used as features, as they contain information produced by interactions in the recommender system.

Chapter 3

Evaluation

This chapter briefly describes the siamese graph neural network implementation and argues for the selected model configuration through rigorous cross-validation testing. It also shows the advantage of a semi-supervised approach to fraud detection and compares the proposed method with existing models.

3.1 Implementation Notes

The proposed siamese graph neural network is implemented in PyTorch[1] with PyTorch Geometric[2]. PyTorch Geometric library provides implementations of graph convolution operators such as SAGEConv, GCNConv, and GATConv. It also includes tools for efficient handling of graph datasets, e.g., indexing and sampling node neighborhoods. The implementation uses neighborhood sampling to scale efficiently to larger graph datasets, which might be impractical to store in memory.

Custom contrastive loss and triplet loss functions are consistent with the descriptions provided by Section 1.4.2. The implementation includes cosine, Euclidean, and SNR-based distance metrics. Pair and triplet sampler implementations sample instances stochastically to reduce computing overhead and provide regularization for the training algorithm by exposing only random subsets of the training data in each epoch.

3.2 Existing Datasets

Datasets used for false review detection must contain more information regarding the interactions than usual recommender system benchmark datasets, which often consist only of the sparse rating matrix and sometimes the user and item attributes. Reviews must be paired with a user (author) and the reviewed item. They typically include a star rating and the text review itself. Additional information may be votes given to the review by other users (agreeing or disagreeing with the review) and metadata such as the date, time, or device from which the review has been sent.

[1] `https://pytorch.org/`

[2] `https://pytorch-geometric.readthedocs.io/en/latest/`

[119] presents a graph dataset where Yelp.com business reviews are represented as nodes and are connected via various types of edges. Each review has 32 handcrafted features (for example, rating deviation from the mean, review text length, or ratio of subjective and objective words). The three types of edges are:

- R-U-R: connecting reviews posted by the same user,
- R-S-R: connecting reviews with the same star rating (1-5 stars) that review the same item,
- R-T-R: connecting reviews posted in the same month for the same item.

In total, there are $45,954$ reviews, $98,630$ R-U-R edges, $1,147,232$ R-T-R edges and $6,805,486$ R-S-R edges. $6,677$ reviews are labeled as spam. This label comes from a proprietary automated system used by Yelp to filter spam reviews. The models using this benchmark aim to learn to imitate the spam detection algorithm.

[120] introduces an alternate graph-based perspective for fraud detection, and instead of reviews, it categorizes users who post reviews (of the items on the e-commerce website Amazon.com). Each user is associated with 25 handcrafted features (e.g., number of reviews, entropy of ratings, average sentiment of text reviews, or ratio of positive and negative ratings). This dataset again contains three types of edges that connect user nodes:

- U-P-U: connecting users that have reviewed the same item,
- U-S-U: connecting users that have given the same (1-5) star rating within one week,
- U-V-U: connecting users who have posted very similar text reviews (top 5% text similarity measured using TF-IDF).

There are $11,944$ users, $351,216$ U-P-U edges, $7,132,958$ U-S-U edges and $2,073,474$ U-V-U edges. 821 users are labeled as fraudulent. Amazon allows other users to rate the reviews as "helpful". In this dataset, users with less than 20% helpful votes are considered fraudsters. The goal is to learn how to predict whether other users will find a user's reviews helpful.

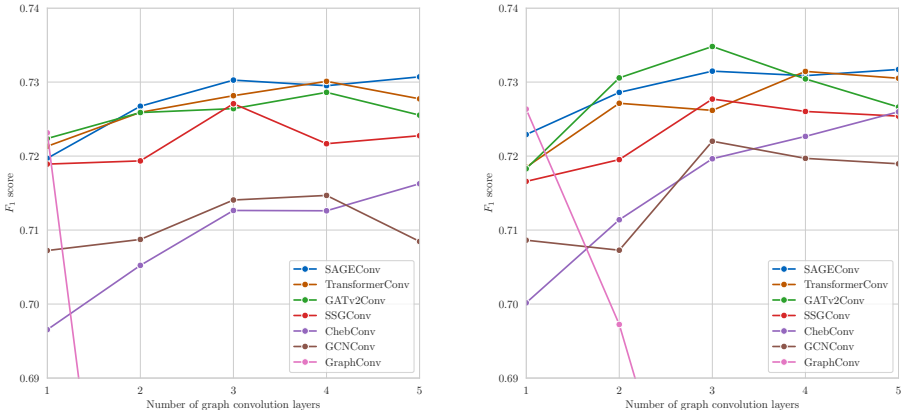
3.3 Selected Hyperparameters

The graph convolution operator used by the CHGC layer and activation functions can be selected based on familiarity; for example, LeakyReLU is often a popular choice. A quick benchmark using the GitHub graph dataset (introduced by [121]), an imbalanced binary node classification task, is performed to make a more informed selection. Nodes represent GitHub users (128 features per node), and edges represent whether a user follows another user. The goal is to classify nodes into two classes: web developers and machine-learning developers.

The performance of different graph convolution operators, the number of convolution layers, and the importance of dropout regularization are illustrated by Figure 3.1 and summarized by Table 3.1. Activation function benchmarks

are done using only a feed-forward neural network, without any graph convolution, increasing the number of hidden layers and dropout regularization.

Figures 3.1, 3.2 and Tables 3.1, 3.2 illustrate the performance of different graph convolution layers and activation functions. Based on these results, SAGEConv is selected as the graph convolution operator used by CHGC, together with the CELU activation function.

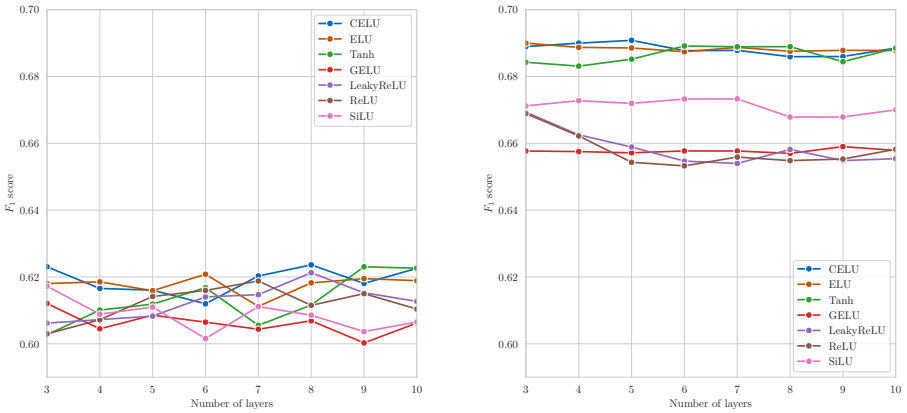


(a) Dropout $p = 0$ (b) Dropout $p = 0.2$

Figure 3.1: Performance comparison of SAGEConv, TransformerConv, GATv2Conv, SSGConv, ChebConv, GCNConv and GraphConv layers. Dropout with given probability p is applied after each layer. Performance is measured using F_1 score for increasing the number of convolutional layers.

Table 3.1: Averaged performance of different convolutional layers for varied number of layers and dropout probability.

Activation function	Average metric performance			
	AUC	F_1 score	AP	G_{mean}
SAGEConv [54]	**0.9056**	**0.7282**	**0.6075**	**0.8248**
TransformerConv [53]	0.9051	0.7267	0.6056	0.8242
GATv2Conv [52]	0.9024	0.7270	0.6057	0.8230
SSGConv [57]	0.9008	0.7225	0.6029	0.8194
ChebConv [55]	0.8908	0.7123	0.5914	0.8107
GCNConv [50]	0.8938	0.7130	0.5933	0.8121
GraphConv [56]	0.7877	0.5931	0.4891	0.7110



(a) Dropout $p = 0$

(b) Dropout $p = 0.2$

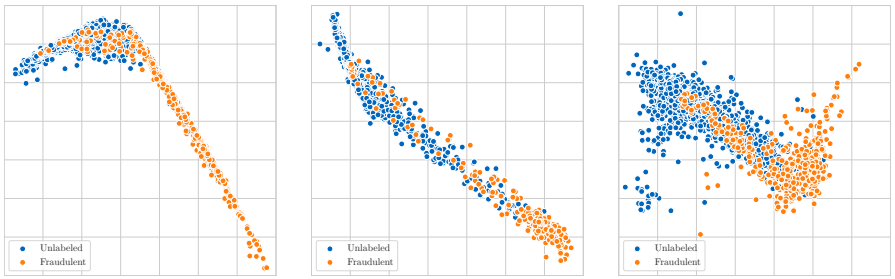
Figure 3.2: Performance comparison of CELU, ELU, Tanh, GELU, LeakyReLU, ReLU, and SiLU activation functions in a feed-forward neural network. Dropout with given probability p is applied after each layer. Performance is measured using F_1 score for increasing number of layers in the feed-forward neural network.

Table 3.2: Averaged performance of different activation functions for varied number of layers and dropout probability.

Activation function	Average metric performance			
	AUC	F_1 score	AP	G_{mean}
CELU	**0.8539**	**0.6606**	**0.5335**	**0.7728**
ELU	0.8534	0.6600	0.5323	0.7724
Tanh	0.8526	0.6565	0.5278	0.7670
GELU	0.8348	0.6406	0.5112	0.7538
LeakyReLU	0.8422	0.6445	0.5140	0.7552
ReLU	0.8418	0.6440	0.5141	0.7547
SiLU	0.8403	0.6478	0.5208	0.7630

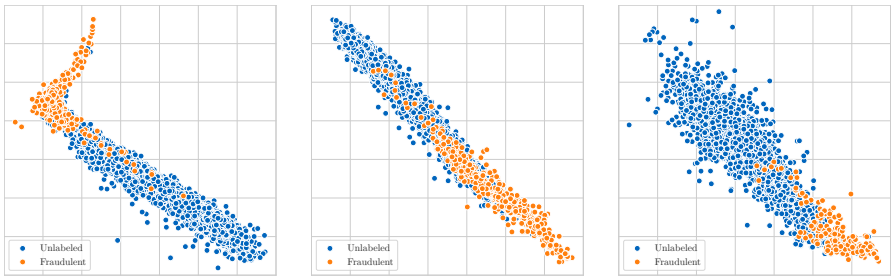
3.3.1 Loss and Distance used by the Decoupled Version

The decoupled siamese graph neural network aims to learn distant latent representations of labeled and unlabeled nodes. Figure 3.3 shows the latent space learned by minimizing contrastive loss using different distance metrics, and Figure 3.4 shows the same learned by minimizing triplet loss.



(a) Cosine distance (b) Euclidean distance (c) SNR-based distance

Figure 3.3: Visualization of node embedding space learned via minimization of contrastive loss with various distance metrics (Amazon dataset). SNR-based distance encourages the model to learn tigher class clusters, while Euclidean distance allows for more spread, and cosine distance enforces learning orthogonal embedding vectors.



(a) Cosine distance (b) Euclidean distance (c) SNR-based distance

Figure 3.4: Visualization of node embedding space learned via minimization of triplet loss with various distance metrics (Amazon dataset). Unlike contrastive loss, the triplet loss encourages less overlap between class clusters through the anchored samples.

The decoupled model measures the distance of node embeddings to determine class affiliation by selecting k random unlabeled samples from the training set to compare against. The final fraud score is calculated as the mean of the k distances. Figure 3.5 visualizes the distribution of these distances, in which it is clear that the unlabeled and labeled (fraudulent) instances follow different distributions. Alternatively, hypothesis testing (as described in Section 2.2.5) can be used to determine which distribution the k distances fit better.

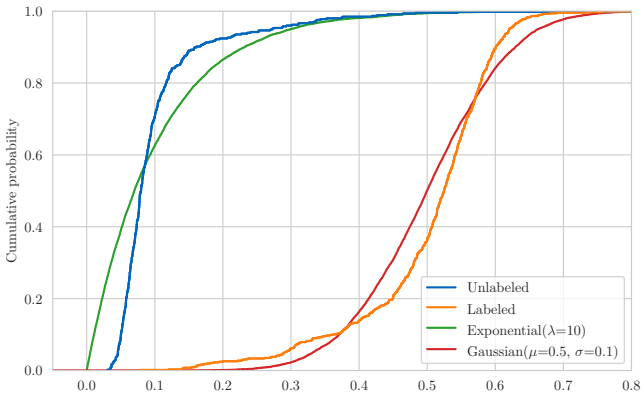


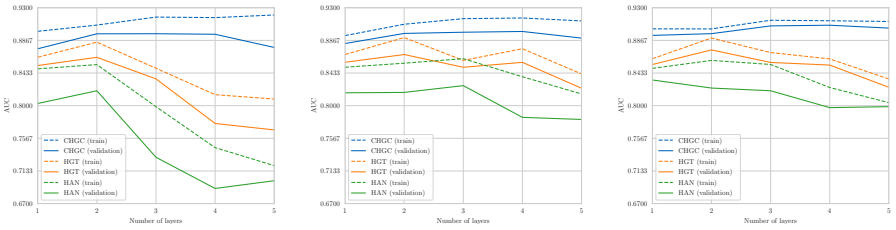
Figure 3.5: Empirical cumulative distribution functions of node embedding distances (average distance from 20 randomly sampled unlabeled instances). The distribution of distances for the unlabeled set resembles an exponential distribution, while the distribution of the distances for the labeled set distances resembles a Gaussian distribution.

3.4 Graph Convolution Depth and Regularization

Adding regularization during training is especially important for the siamese graph neural network, as the dual component ϕ_1 uses shared weights and thus is exposed to twice as many training examples. The sample pairing only exacerbates the problem via the natural data augmentation from m individual samples to m^2 possible pairs of samples, making the model prone to overfitting.

In this experiment, two different existing heterogeneous graph convolution operators (HGT [113] and HAN [115]) are compared with the CHGC operator proposed in Section 2.2.2. These operators sit in place of the ϕ_1 function, which creates the node neighborhood embeddings.

The effect of increasing the number of graph convolution layers and dropout regularization is shown in Figure 3.6 and summarized in Table 3.3 for the Yelp dataset. CHGC shows a significant advantage over both HGT and HAN at every depth. Higher dropout probability also provides resistance to overfitting.



(a) Dropout $p = 0$ (b) Dropout $p = 0.1$ (c) Dropout $p = 0.2$

Figure 3.6: Performance of different heterogeneous graph convolution operators for an increasing number of layers and varied dropout probability on the Yelp dataset.

Table 3.3: AUC performance for HGT, HAN, and CHGC using different dropout probability during training and increasing number of layers on the Yelp dataset. A higher number of graph convolution layers does not necessarily lead to better performance, as aggregating information from many distant nodes can result in dilution, i.e., all nodes are represented by a similar vector.

Heterogeneous convolution	Dropout p	Number of layers				
		1	2	3	4	5
HGT [113]	0.0	0.8643	0.8843	0.8356	0.7764	0.7678
	0.1	0.8576	0.8679	0.8510	0.8575	0.8231
	0.2	0.8546	0.8740	0.8574	0.8539	0.8244
HAN [115]	0.0	0.8029	0.8198	0.7315	0.6901	0.7004
	0.1	0.8171	0.8176	0.8266	0.7845	0.7817
	0.2	0.8341	0.8234	0.8198	0.8774	0.7987
CHGC	0.0	0.8755	0.8954	0.8955	0.8948	0.8773
	0.1	0.8825	**0.8959**	0.8975	0.8985	0.8897
	0.2	**0.8933**	0.8956	**0.9059**	**0.9068**	**0.9030**

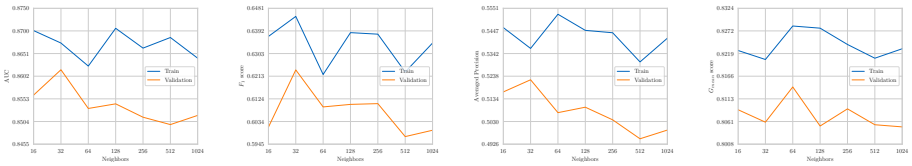
3.5 Neighborhood Size

The graph neural network creates embeddings of node neighborhoods. On average, every node $\mathbf{v} \in \mathcal{V}$ from graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ has $1,600$ neighbors in the Amazon dataset and 350 neighbors in the Yelp dataset. The node neighborhoods are randomly undersampled as a way of regularization (hiding information) and data augmentation (unique subgraph in every training epoch). The size of the sampled neighborhood is a hyperparameter, the effect of which is explored in the following experiment.

The siamese graph neural network is trained using varied sizes of neighborhoods, evaluating performance benefits, as illustrated in Figure 3.7 for the Amazon dataset, Figure 3.8 for the Yelp dataset, and both are summarized in Table 3.4.

(a) AUC (b) F_1 score (c) AP (d) G_{mean} score

Figure 3.7: Evaluation metrics for increasing size of node neighborhoods on the Amazon dataset. For each metric, there is a slight decreasing trend, suggesting that sampling smaller neighborhoods is beneficial, akin to regularization.



(a) AUC (b) F_1 score (c) AP (d) G_{mean} score

Figure 3.8: Evaluation metrics for varied size of node neighborhoods on the Yelp dataset.

Table 3.4: Evaluated metrics for varied size of node neighborhoods for Amazon and Yelp datasets.

Neighbors	Amazon				Yelp			
	AUC	F_1 score	AP	G_{mean}	AUC	F_1 score	AP	G_{mean}
16	0.9761	0.8670	0.7775	0.9493	0.8701	0.6370	0.5462	0.8226
32	**0.9779**	0.8820	0.7896	0.9490	0.8674	0.**6449**	0.5366	0.8205
64	0.9662	**0.8906**	**0.8084**	**0.9604**	0.8624	0.6220	**0.5523**	**0.8283**
128	0.9676	0.8725	0.7604	0.9421	**0.8706**	0.6385	0.5449	0.8278
256	0.9750	0.8674	0.7636	0.9496	0.8663	0.6379	0.5438	0.8240
512	0.9646	0.8463	0.7794	0.9465	0.8686	0.6230	0.5304	0.8208
1024	0.9713	0.8728	0.7675	0.9455	0.8641	0.6344	0.5413	0.8230

3.5.1 Neighborhood Resampling

To explore the effect of random neighborhood subgraphs, the variance of predictions can be examined. In this experiment, a siamese graph neural network ϕ is trained to output a fraud score for each node (higher for fraudulent nodes, lower for legitimate nodes). For every node $\mathbf{v}$, its neighborhood is sampled randomly 20 times, creating 20 unique outputs of the network $\mathbf{y}_i = \phi(\mathcal{G}_\mathbf{v}; \theta)$.

The variance of the random vector $\mathbf{y}$ is calculated as $s^2_\mathbf{v} = \frac{\sum_{i=1}^{20}(\mathbf{y}_i - \bar{\mathbf{y}})}{19}$ and the distribution of these variances (one for each node $\mathbf{v} \in \mathcal{V}$) is plotted in Figure 3.9. As the figure shows, variance for each class never exceeds 10^{-2}, with the most common variance values falling in the range between 10^{-6} and 10^{-8}. The outputted scores are in the $[0, 2]$ range, meaning a 10^{-7} variance due to a randomly sampled node neighborhood has minimal impact on the final fraud score.

3.6 Number of Pairs per Prediction

The fraud score for a given node is calculated as the mean of $k \in \mathbb{N}$ pairwise predictions. Larger k should improve resistance against fraud contamination within the unlabeled instances; as per the law of large numbers, with a growing sample size, the average converges to the true value. This experiment evaluates the performance benefits of using a larger number of paired instances per prediction.

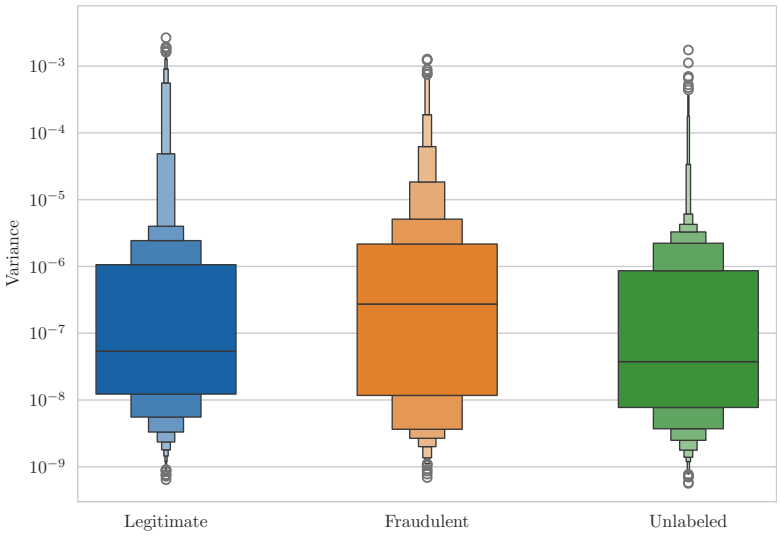
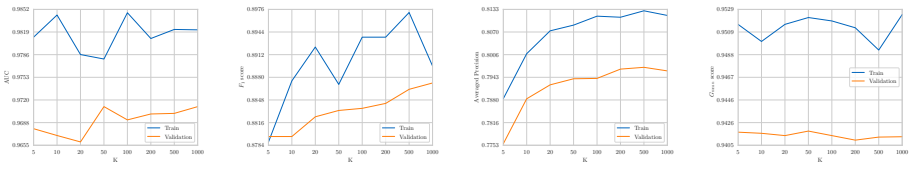
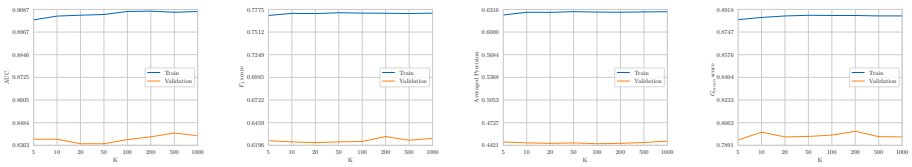


Figure 3.9: Letter-value plot [122] of fraud score variances for randomly sampled node neighborhoods. Legitimate, fraudulent, and unlabeled are the ground truth labels of instances. These classes are plotted separately for introspection into their respective variance distributions.



(a) AUC (b) F_1 score (c) AP (d) G_{mean} score

Figure 3.10: Evaluation metrics for increasing number of pairs on the Amazon dataset. There is an evident upward trend, suggesting that a larger number of pairs can lead to better performance.



(a) AUC (b) F_1 score (c) AP (d) G_{mean} score

Figure 3.11: Evaluation metrics for increasing number of pairs on the Yelp dataset.

Table 3.5: Evaluated metrics for increasing number of pairs per prediction for Amazon and Yelp datasets.

Pairs	Amazon				Yelp			
	AUC	F_1 score	AP	G_{mean}	AUC	F_1 score	AP	G_{mean}
5	0.9679	0.8796	0.7883	0.9417	0.9032	0.7704	0.6238	0.8841
10	0.9669	0.8796	0.7883	0.9416	0.9052	0.7729	0.6276	0.8857
20	0.9659	0.8824	0.7922	0.9414	0.9056	0.7727	0.6275	0.8868
50	**0.9711**	0.8833	0.7939	**0.9418**	0.9060	**0.7736**	0.6284	**0.8874**
100	0.9691	0.8836	0.7940	0.9414	0.9077	0.7731	0.6279	0.8872
200	0.9700	0.8843	0.7966	0.9410	**0.9078**	0.7730	0.6277	0.8872
500	0.9701	0.8863	0.7991	0.9412	0.9072	0.7727	0.6281	0.8869
1000	**0.9711**	**0.8872**	**0.8034**	0.9413	0.9076	0.7731	**0.6284**	0.8869

3.7 Robustness Against Contamination

In a real-world scenario, where a set of m users contains $f \ll m$ fraudsters, and $|\mathcal{A}| = p$ of them are known, there remain $f - p$ camouflaged fraudsters. The known set of fraudsters $\mathcal{A}$ is used to train the fraud detection model in the semi-supervised setting. The proposed siamese graph neural network also utilizes the unlabeled set of $|\mathcal{N}| = m - p$ instances to learn fraud representations. This experiment evaluates the resilience of the siamese graph neural network against camouflaged fraudsters by varying the size of the labeled set $\mathcal{A}$. Both coupled and decoupled models are compared against a traditional multi-layer perceptron (MLP) classifier, trained to minimize the binary cross entropy (BCE) loss:

$$\mathcal{L}_{BCE}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{m} \sum_{i=1}^{m} \left(\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i) \right),$$

where $\mathbf{y} \in \{0, 1\}^m$ is the target and $\hat{\mathbf{y}} \in [0, 1]^m$ is the predicted probability of the node being fraudulent. As is common in binary classification tasks, the last activation function of the classifier is the sigmoid function.

Table 3.6 summarizes the results, showing that the proposed siamese graph neural network with a coupled fraud scorer function is more resilient to class contamination than a traditional end-to-end classifier. It performs better even in the case of zero contamination (i.e., a supervised task), thanks to the graph convolution layers, which allow it to exploit graph structure features, unlike a traditional multi-layer perceptron.

3.8 Comparison with Existing Methods

This section compares the performance of the proposed siamese graph neural network against existing baselines that use the Amazon and Yelp datasets. For each model, the dataset is split into 60/20/20 training/validation/testing sets, and the best hyperparameters are selected based on the performance on the validation set; then, the model is retrained using instances from both training and validation before evaluating performance on the testing set. Table 3.7

Table 3.6: AUC performance with camouflaged fraudsters (fraud percentage in the unlabeled set). Amazon, with 6% contamination, offers only 67 known instances of fraud (against 7,098 unlabeled instances). Yelp with 14% contamination, the numbers are 170 fraudsters and 27,402 unlabeled.

Model	Amazon				Yelp			
	0%	2%	4%	6%	0%	5%	10%	14%
Siamese GNN (coupled)	**0.9793**	**0.9763**	**0.9768**	**0.9625**	**0.9112**	**0.9059**	**0.8966**	**0.7661**
Siamese GNN (decoupled)	0.9534	0.9519	0.9497	0.9434	0.8411	0.8295	0.8282	0.7412
MLP	0.9743	0.9688	0.9606	0.9518	0.8144	0.8183	0.8009	0.7420

shows the comparison for seven different baselines and four different fraud detection graph datasets using the AUC metric. AUC was chosen as it is used by all the mentioned papers.

Amazon and Yelp datasets have been described previously. FDCompCN (introduced by [100]) is a financial fraud dataset that represents a number of companies in China. It contains 5,317 nodes, 559 of which are labeled fraudulent. Each node is represented by 57 features (such as the number of employees, industry, company type, registered capital, or city), and there are three types of relationships connecting them:

- C-C-C: connecting companies that have an investment relationship,
- C-P-C: connecting companies based on shared customers,
- C-S-C: connecting companies that use the same supplier(s).

S-FFSD is a synthetic semi-supervised financial fraud dataset introduced by [101]. However, unlike the other graph datasets, it is homogeneous (contains only one type of node and one type of relationship). In S-FFSD, there are 77,881 nodes and 335,139 edges. Each node has 126 features, 5,256 are labeled as fraudulent, 24,387 as legitimate, and 48,238 are unlabeled.

Table 3.7: Performance comparison with existing baseline methods. The table lists only the AUC metric, which is shared between all the respective papers.

Model	Dataset			
	Amazon	Yelp	FDCompCN	S-FFSD
Siamese GNN (coupled)	**0.9815**	0.9221	**0.7733**	**0.8327**
Siamese GNN (decoupled)	0.9413	0.8862	0.6726	0.7555
CARE-GNN [96]	0.8973	0.7570	0.6518	0.6623
SplitGNN [100]	0.9323	0.9203	0.6898	–
GTAN [101]	0.9630	**0.9241**	–	0.7616
PC-GNN [97]	0.9586	0.7987	–	0.6795
RioGNN [98]	0.9403	0.8238	0.6347	–
DCGNN [102]	0.9758	0.9085	–	–
RLC-GNN [99]	0.9748	0.8544	–	–

Conclusion

The goal of this thesis was to design and implement a novel graph-based semi-supervised approach for detecting fraud in recommender systems. Fraudsters typically attempt to camouflage their behavior in order to remain undetected, which makes creating a quality dataset challenging. In addition, fraudsters represent only a minority of instances, leading to imbalanced class sizes even if labeled correctly.

Previous research has shown that taking advantage of relationships between instances can offer better detection capabilities. This work proposed a new heterogeneous graph convolution operator and two versions of a siamese graph neural network, together which have demonstrated their ability to match, and in some cases even outperform, existing graph-based models (as shown in Section 3.8, where it is compared with seven baseline models on four different datasets).

The proposed siamese network is trained in a unique way, requiring new paired node neighborhood data loaders, the implementation of which is provided, together with both coupled and decoupled versions of the model. Both versions analyze pairs of instances, leveraging the larger set of unlabeled instances. The fraudster minority assumption also makes it more resistant to contamination within the unlabeled set, unlike traditional binary classifiers (demonstrated in Section 3.7).

Further experiments have also evaluated hyperparameter selection, such as the number of graph convolution layers, activation functions, dropout regularization, loss and distance functions, the size of the sampled node neighborhood, and the number of pairs per prediction.

Interpretable aspects of the siamese graph neural network have been discussed in Section 2.2.5, where hypothesis testing is proposed as a way to calculate confidence in the prediction of the model by comparing the distribution of embedding distances. Additionally, GraphLIME was mentioned as it can also offer an explanation of the most representative features of nodes in a graph neighborhood.

Future work direction entails replacing the twin graph neural network modules inside the siamese network with existing, pre-trained models with missing heads. This straightforward transfer learning procedure could allow underperforming models to get a boost in performance by taking advantage of its resistance to class contamination.

Bibliography

1. SCHWARTZ, Barry. *The paradox of choice: Why more is less*. HarperCollins Publishers, 2004. ISBN 0-06-000568-8.
2. CHERNEV, Alexander; BÖCKENHOLT, Ulf; GOODMAN, Joseph. Choice overload: A conceptual review and meta-analysis. *Journal of Consumer Psychology*. 2015, vol. 25, no. 2, pp. 333–358. Available from DOI: 10.1016/j.jcps.2014.08.002.
3. AGARWAL, Vinti; BHARADWAJ, K. K. A collaborative filtering framework for friends recommendation in social networks based on interaction intensity and adaptive user similarity. *Social Network Analysis and Mining*. 2013, vol. 3, no. 3, pp. 359–379. ISSN 1869-5469. Available from DOI: 10.1007/s13278-012-0083-7.
4. GOMEZ-URIBE, Carlos A.; HUNT, Neil. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 2016, vol. 6, no. 4. ISSN 2158-656X. Available from DOI: 10.1145/2843948.
5. KARIMI, Mozhgan; JANNACH, Dietmar; JUGOVAC, Michael. News recommender systems - Survey and roads ahead. *Information Processing & Management*. 2018, vol. 54, no. 6, pp. 1203–1227. ISSN 0306-4573. Available from DOI: 10.1016/j.ipm.2018.04.008.
6. BEEL, Joeran; GIPP, Bela; LANGER, Stefan; BREITINGER, Corinna. Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*. 2016, vol. 17, no. 4, pp. 305–338. ISSN 1432-1300. Available from DOI: 10.1007/s00799-015-0156-0.
7. LI, Seth Siyuan; KARAHANNA, Elena. Online Recommendation Systems in a B2C E-Commerce Context: A Review and Future Directions. *Journal of the Association for Information Systems*. 2015, vol. 16, no. 2. Available from DOI: 10.17705/1jais.00389.
8. VARGAS GOVEA, Blanca; SERNA, Gabriel; PONCE-MEDELLÍN, Rafael. Effects of relevant contextual features in the performance of a restaurant recommender system. In: 2011, vol. 791.

9. LEVI, Asher; MOKRYN, Osnat; DIOT, Christophe; TAFT, Nina. Finding a needle in a haystack of reviews: cold start context-based hotel recommender system. In: Dublin, Ireland: Association for Computing Machinery, 2012, pp. 115–122. RecSys '12. ISBN 9781450312707. Available from DOI: 10.1145/2365952.2365977.

10. DE CROON, Robin; VAN HOUDT, Leen; HTUN, Nyi Nyi; ŠTIGLIC, Gregor; VANDEN ABEELE, Vero; VERBERT, Katrien. Health Recommender Systems: Systematic Review. *J Med Internet Res.* 2021, vol. 23, no. 6, e18035. ISSN 1438-8871. Available from DOI: 10.2196/18035.

11. JANNACH, Dietmar; JUGOVAC, Michael. Measuring the Business Value of Recommender Systems. 2019, vol. 10, no. 4. ISSN 2158-656X. Available from DOI: 10.1145/3370082.

12. LAM, Shyong K.; RIEDL, John. Shilling recommender systems for fun and profit. In: *Proceedings of the 13th International Conference on World Wide Web.* New York, NY, USA: Association for Computing Machinery, 2004, pp. 393–402. WWW '04. ISBN 158113844X. Available from DOI: 10.1145/988672.988726.

13. GUNES, Ihsan; KALELI, Cihan; BILGE, Alper; POLAT, Huseyin. Shilling attacks against recommender systems: a comprehensive survey. *Artificial Intelligence Review.* 2014, vol. 42, no. 4, pp. 767–799. ISSN 1573-7462. Available from DOI: 10.1007/s10462-012-9364-9.

14. ADOMAVICIUS, Gediminas; ZHANG, Jingjing. Impact of data characteristics on recommender systems performance. *ACM Trans. Manage. Inf. Syst.* 2012, vol. 3, no. 1. ISSN 2158-656X. Available from DOI: 10.1145/2151163.2151166.

15. SAR SHALOM, Oren; BERKOVSKY, Shlomo; RONEN, Royi; ZIKLIK, Elad; AMIHOOD, Amir. Data Quality Matters in Recommender Systems. In: Vienna, Austria: Association for Computing Machinery, 2015, pp. 257–260. RecSys '15. ISBN 9781450336925. Available from DOI: 10.1145/2792838.2799670.

16. MEWADA, Arvind; DEWANG, Rupesh Kumar. Research on false review detection Methods: A state-of-the-art review. *Journal of King Saud University - Computer and Information Sciences.* 2022, vol. 34, no. 9, pp. 7530–7546. ISSN 1319-1578. Available from DOI: 10.1016/j.jksuci.2021.07.021.

17. PRAMITHA, Febriora Nevia; HADIPRAKOSO, Raden Budiarto; QOMARIASIH, Nurul; GIRINOTO. Twitter Bot Account Detection Using Supervised Machine Learning. In: *2021 4th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI).* 2021, pp. 379–383. Available from DOI: 10.1109/ISRITI54043.2021.9702789.

18. WIMMER, Hayden; YOON, Victoria Y. Counterfeit product detection: Bridging the gap between design science and behavioral science in information systems research. *Decision Support Systems.* 2017, vol. 104, pp. 1–12. ISSN 0167-9236. Available from DOI: 10.1016/j.dss.2017.09.005.

19. ZHANG, Fuguo. A Survey of Shilling Attacks in Collaborative Filtering Recommender Systems. In: *2009 International Conference on Computational Intelligence and Software Engineering*. 2009, pp. 1–4. Available from DOI: 10.1109/CISE.2009.5365077.

20. WU, Zhiang; WU, Junjie; CAO, Jie; TAO, Dacheng. HySAD: a semi-supervised hybrid shilling attack detector for trustworthy product recommendation. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Beijing, China: Association for Computing Machinery, 2012, pp. 985–993. KDD '12. ISBN 9781450314626. Available from DOI: 10.1145/2339530.2339684.

21. YU, Hongtao; SUN, Lijun; AND, Fuzhi Zhang. A Robust Bayesian Probabilistic Matrix Factorization Model for Collaborative Filtering Recommender Systems Based on User Anomaly Rating Behavior Detection. *KSII Transactions on Internet and Information Systems*. 2019, vol. 13, no. 9, pp. 4684–4705. Available from DOI: 10.3837/tiis.2019.09.020.

22. ANELLI, Vito Walter; DELDJOO, Yashar; DI NOIA, Tommaso; DI SCIASCIO, Eugenio; MERRA, Felice Antonio. SAShA: Semantic-Aware Shilling Attacks on Recommender Systems Exploiting Knowledge Graphs. In: *The Semantic Web*. Cham: Springer International Publishing, 2020, pp. 307–323. ISBN 978-3-030-49461-2. Available from DOI: 10.1007/978-3-030-49461-2_18.

23. YUAN, Liu; HONGMIN, Chen; RUIKUN, Duan. Whom to benefit? Competing platforms' strategic investment in recommender systems. *Electronic Commerce Research and Applications*. 2022, vol. 56, p. 101210. ISSN 1567-4223. Available from DOI: 10.1016/j.elerap.2022.101210.

24. SULIKOWSKI, Piotr; ZDZIEBKO, Tomasz; TURZYŃSKI, Dominik; KAŃTOCH, Eliasz. Human-website interaction monitoring in recommender systems. *Procedia Computer Science*. 2018, vol. 126, pp. 1587–1596. ISSN 1877-0509. Available from DOI: 10.1016/j.procs.2018.08.132. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia.

25. CHEN, Yen-Liang; WENG, Cheng-Hsiung. Mining fuzzy association rules from questionnaire data. *Knowledge-Based Systems*. 2009, vol. 22, no. 1, pp. 46–56. ISSN 0950-7051. Available from DOI: 10.1016/j.knosys.2008.06.003.

26. MADUAKO, Iyke; GONG, Yaqi; WACHOWICZ, Monica. Building k-partite association graphs for finding recommendation patterns from questionnaire data. *Transactions in GIS*. 2021, vol. 25, no. 5, pp. 2641–2659. Available from DOI: 10.1111/tgis.12787.

27. AGUIAR, Luis; PEUKERT, Christian; SCHÄFER, Maximilian; ULLRICH, Hannes. *Facebook Shadow Profiles*. 2022. Available from DOI: 10.48550/arXiv.2202.04131.

28. HIMEUR, Yassine; SOHAIL, Shahab Saquib; BENSAALI, Faycal; AMIRA, Abbes; ALAZAB, Mamoun. Latest trends of security and privacy in recommender systems: A comprehensive review and future perspectives. *Computers & Security*. 2022, vol. 118, p. 102746. ISSN 0167-4048. Available from DOI: 10.1016/j.cose.2022.102746.

29. UR, Blase; LEON, Pedro Giovanni; CRANOR, Lorrie Faith; SHAY, Richard; WANG, Yang. Smart, useful, scary, creepy: perceptions of online behavioral advertising. In: Washington, D.C.: Association for Computing Machinery, 2012. SOUPS '12. ISBN 9781450315326. Available from DOI: 10.1145/2335356.2335362.

30. LEON, Pedro; UR, Blase; WANG, Yang; SLEEPER, Manya; BALEBAKO, Rebecca; SHAY, Richard; BAUER, Lujo; CHRISTODORESCU, Mihai; CRANOR, Lorrie. What Matters to Users? Factors that Affect Users' Willingness to Share Information with Online Advertisers. In: 2013. Available from DOI: 10.1145/2501604.2501611.

31. SABRINA KARWATZKI Olga Dytynko, Manuel Trenz; VEIT, Daniel. Beyond the Personalization–Privacy Paradox: Privacy Valuation, Transparency Features, and Service Personalization. *Journal of Management Information Systems*. 2017, vol. 34, no. 2, pp. 369–400. Available from DOI: 10.1080/07421222.2017.1334467.

32. ADOMAVICIUS, G.; TUZHILIN, A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*. 2005, vol. 17, no. 6, pp. 734–749. Available from DOI: 10.1109/TKDE.2005.99.

33. ARMEN S. ASRATIAN Tristan M. J. Denley, Roland Häggkvist. *Bipartite graphs and their applications*. Cambridge University Press, 1998. Cambridge Tracts in Mathematics. ISBN 052159345X, ISBN 9780521593458.

34. AREEB, Qazi Mohammad; NADEEM, Mohammad; SOHAIL, Shahab Saquib; IMAM, Raza; DOCTOR, Faiyaz; HIMEUR, Yassine; HUSSAIN, Amir; AMIRA, Abbes. Filter bubbles in recommender systems: Fact or fallacy - A systematic review. *WIREs Data Mining and Knowledge Discovery*. 2023, vol. 13, no. 6, e1512. Available from DOI: 10.1002/widm.1512.

35. KASALICKY, Petr; LEDENT, Antoine; ALVES, Rodrigo. Uncertainty-adjusted Inductive Matrix Completion with Graph Neural Networks. In: *Proceedings of the 17th ACM Conference on Recommender Systems*. Singapore, Singapore: Association for Computing Machinery, 2023, pp. 1169–1174. RecSys '23. ISBN 9798400702419. Available from DOI: 10.1145/3604915.3610654.

36. WANG, Bin; LIAO, Qing; ZHANG, Chunhong. Weight Based KNN Recommender System. In: *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*. 2013, vol. 2, pp. 449–452. Available from DOI: 10.1109/IHMSC.2013.254.

37. RENDLE, Steffen; FREUDENTHALER, Christoph; GANTNER, Zeno; SCHMIDT-THIEME, Lars. *BPR: Bayesian Personalized Ranking from Implicit Feedback*. 2012. Available from DOI: 10.48550/arXiv.1205.2618.

38. JOHNSON, Christopher C. Logistic Matrix Factorization for Implicit Feedback Data. *Advances in Neural Information Processing Systems*. 2014, vol. 27, no. 78, pp. 1–9. Available also from: https://web.stanford.edu/~rezab/nips2014workshop/submits/logmat.pdf.

39. DUCHI, John; HAZAN, Elad; SINGER, Yoram. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*. 2011, vol. 12, pp. 2121–2159. ISSN 1532-4435.

40. ZEILER, Matthew D. *ADADELTA: An Adaptive Learning Rate Method*. 2012. Available from DOI: 10.48550/arXiv.1212.5701.

41. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. Available from DOI: 10.48550/arXiv.1412.6980.

42. LOSHCHILOV, Ilya; HUTTER, Frank. *Decoupled Weight Decay Regularization*. 2019. Available from DOI: 10.48550/arXiv.1711.05101.

43. HORNIK, Kurt; STINCHCOMBE, Maxwell; WHITE, Halbert. Multilayer feedforward networks are universal approximators. *Neural Networks*. 1989, vol. 2, no. 5, pp. 359–366. ISSN 0893-6080. Available from DOI: 10.1016/0893-6080(89)90020-8.

44. SEDHAIN, Suvash; MENON, Aditya Krishna; SANNER, Scott; XIE, Lexing. AutoRec: Autoencoders Meet Collaborative Filtering. In: *Proceedings of the 24th International Conference on World Wide Web*. Florence, Italy: Association for Computing Machinery, 2015, pp. 111–112. WWW '15 Companion. ISBN 9781450334730. Available from DOI: 10.1145/2740908.2742726.

45. ZHANG, Guijuan; LIU, Yang; JIN, Xiaoning. A survey of autoencoder-based recommender systems. *Frontiers of Computer Science*. 2020, vol. 14, no. 2, pp. 430–450. ISSN 2095-2236. Available from DOI: 10.1007/s11704-018-8052-6.

46. SCARSELLI, Franco; GORI, Marco; TSOI, Ah Chung; HAGENBUCHNER, Markus; MONFARDINI, Gabriele. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*. 2009, vol. 20, no. 1, pp. 61–80. Available from DOI: 10.1109/TNN.2008.2005605.

47. WU, Shiwen; SUN, Fei; ZHANG, Wentao; XIE, Xu; CUI, Bin. Graph Neural Networks in Recommender Systems: A Survey. 2022, vol. 55, no. 5. ISSN 0360-0300. Available from DOI: 10.1145/3535101.

48. GAO, Chen; ZHENG, Yu; LI, Nian; LI, Yinfeng; QIN, Yingrong; PIAO, Jinghua; QUAN, Yuhan; CHANG, Jianxin; JIN, Depeng; HE, Xiangnan; LI, Yong. A Survey of Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions. 2023, vol. 1, no. 1. Available from DOI: 10.1145/3568022.

49. GIRSHICK, Ross; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. Available from DOI: 10.48550/arXiv.1311.2524.

50. KIPF, Thomas N.; WELLING, Max. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. Available from DOI: 10.48550/arXiv.1609.02907.

51. VELIČKOVIĆ, Petar; CUCURULL, Guillem; CASANOVA, Arantxa; ROMERO, Adriana; LIÒ, Pietro; BENGIO, Yoshua. *Graph Attention Networks*. 2018. Available from DOI: 10.48550/arXiv.1710.10903.

52. BRODY, Shaked; ALON, Uri; YAHAV, Eran. *How Attentive are Graph Attention Networks?* 2022. Available from DOI: 10.48550/arXiv.2105.14491.

53. SHI, Yunsheng; HUANG, Zhengjie; FENG, Shikun; ZHONG, Hui; WANG, Wenjin; SUN, Yu. *Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification*. 2021. Available from DOI: 10.48550/arXiv.2009.03509.

54. HAMILTON, William L.; YING, Rex; LESKOVEC, Jure. *Inductive Representation Learning on Large Graphs*. 2018. Available from DOI: 10.48550/arXiv.1706.02216.

55. DEFFERRARD, Michaël; BRESSON, Xavier; VANDERGHEYNST, Pierre. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. 2017. Available from DOI: 10.48550/arXiv.1606.09375.

56. MORRIS, Christopher; RITZERT, Martin; FEY, Matthias; HAMILTON, William L.; LENSSEN, Jan Eric; RATTAN, Gaurav; GROHE, Martin. *Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks*. 2021. Available from DOI: 10.48550/arXiv.1810.02244.

57. ZHU, Hao; KONIUSZ, Piotr. Simple Spectral Graph Convolution. In: *International Conference on Learning Representations*. 2021. Available also from: https://openreview.net/forum?id=CYO5T-YjWZV.

58. ABADAL, Sergi; JAIN, Akshay; GUIRADO, Robert; LÓPEZ-ALONSO, Jorge; ALARCÓN, Eduard. *Computing Graph Neural Networks: A Survey from Algorithms to Accelerators*. 2021. Available from DOI: 10.48550/arXiv.2010.00130.

59. WU, Zonghan; PAN, Shirui; CHEN, Fengwen; LONG, Guodong; ZHANG, Chengqi; YU, Philip S. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*. 2021, vol. 32, no. 1, pp. 4–24. ISSN 2162-2388. Available from DOI: 10.1109/tnnls.2020.2978386.

60. ZHANG, Ziwei; CUI, Peng; ZHU, Wenwu. *Deep Learning on Graphs: A Survey*. 2020. Available from DOI: 10.48550/arXiv.1812.04202.

61. LIANG, Fan; QIAN, Cheng; YU, Wei; GRIFFITH, David; GOLMIE, Nada. Survey of Graph Neural Networks and Applications. *Wireless Communications and Mobile Computing*. 2022, vol. 2022, p. 9261537. ISSN 1530-8669. Available from DOI: 10.1155/2022/9261537.

62. MCKEEVER, Grinne. Detecting, Prosecuting and Punishing Benefit Fraud: The Social Security Administration (Fraud) Act 1997. *The Modern Law Review*. 1999, vol. 62, no. 2, pp. 261–270. Available from DOI: 10.1111/1468-2230.00204.

63. SINGH, Ajeet; JAIN, Anurag. An Empirical Study of AML Approach for Credit Card Fraud Detection–Financial Transactions. *International Journal of Computers Communications & Control*. 2019, vol. 14, pp. 670–690. Available from DOI: 10.15837/ijccc.2019.6.3498.

64. WARREN, Danielle E.; SCHWEITZER, Maurice E. When Lying Does Not Pay: How Experts Detect Insurance Fraud. *Journal of Business Ethics*. 2018, vol. 150, no. 3, pp. 711–726. ISSN 1573-0697. Available from DOI: 10.1007/s10551-016-3124-8.

65. ZHAN, Qing; YIN, Hang. A loan application fraud detection method based on knowledge graph and neural network. In: Shanghai, China: Association for Computing Machinery, 2018, pp. 111–115. ICIAI '18. ISBN 9781450363457. Available from DOI: 10.1145/3194206.3194208.

66. HANCU-BUDUI, Andreea; ZORIO-GRIMA, Ana; BLANCO-VEGA, Jose. The quest for legitimacy: The European Court of Auditors' work on fraud. *Financial Accountability & Management*. 2023. Available from DOI: 10.1111/faam.12375.

67. TROZZE, Arianna; KAMPS, Josh; AKARTUNA, Eray Arda; HETZEL, Florian J.; KLEINBERG, Bennett; DAVIES, Toby; JOHNSON, Shane D. Cryptocurrencies and future financial crime. *Crime Science*. 2022, vol. 11, no. 1. ISSN 2193-7680. Available from DOI: 10.1186/s40163-021-00163-8.

68. ROUKEMA, Boudewijn F. A first-digit anomaly in the 2009 Iranian presidential election. *Journal of Applied Statistics*. 2014, vol. 41, no. 1, pp. 164–199. Available from DOI: 10.1080/02664763.2013.838664.

69. YU, Frank; YU, Xiaoyun. Corporate Lobbying and Fraud Detection. *Journal of Financial and Quantitative Analysis*. 2011, vol. 46, no. 6, pp. 1865–1891. Available from DOI: 10.1017/S0022109011000457.

70. TACKETT, James A. Bribery and corruption. *Journal of Corporate Accounting & Finance*. 2010, vol. 21, no. 4, pp. 5–9. Available from DOI: 10.1002/jcaf.20589.

71. BARSON, P.; FIELD, Simon; DAVEY, N.; MCASKIE, Gill; FRANK, R. Detection of fraud in mobile phone networks. 1996, vol. 6, pp. 477–484.

72. CHADYŠAS, Viktoras; BUGAJEV, Andrej; KRIAUZIENĖ, Rima; VASILECAS, Olegas. Outlier Analysis for Telecom Fraud Detection. In: *Digital Business and Intelligent Systems*. Cham: Springer International Publishing, 2022, pp. 219–231. ISBN 978-3-031-09850-5. Available from DOI: 10.1007/978-3-031-09850-5_15.

73. EL DARRA, Nada; RAJHA, Hiba N.; SALEH, Fatima; AL-OWEINI, Rami; MAROUN, Richard G.; LOUKA, Nicolas. Food fraud detection in commercial pomegranate molasses syrups by UV–VIS spectroscopy, ATR-FTIR spectroscopy and HPLC methods. *Food Control*. 2017, vol. 78, pp. 132–137. ISSN 0956-7135. Available from DOI: 10.1016/j.foodcont.2017.02.043.

74. AGHILI, Nadia Sadat; RASEKH, Mansour; KARAMI, Hamed; AZIZI, Vahid; GANCARZ, Marek. Detection of fraud in sesame oil with the help of artificial intelligence combined with chemometrics methods and chemical compounds characterization by gas chromatography–mass spectrometry. *LWT - Food Science and Technology*. 2022, vol. 167. ISSN 0023-6438. Available from DOI: 10.1016/j.lwt.2022.113863.

75. PUERTAS, Gema; VÁZQUEZ, Manuel. Fraud detection in hen housing system declared on the eggs' label: An accuracy method based on UV-VIS-NIR spectroscopy and chemometrics. *Food Chemistry*. 2019, vol. 288, pp. 8–14. ISSN 0308-8146. Available from DOI: 10.1016/j.foodchem.2019.02.106.

76. HANS-PETER; KROGER, Peer; SCHUBERT, Erich; ZIMEK, Arthur. Interpreting and Unifying Outlier Scores. In: *Proceedings of the 2011 SIAM International Conference on Data Mining (SDM)*. 2011, pp. 13–24. ISBN 978-0-89871-992-5. Available from DOI: 10.1137/1.9781611972818.2.

77. SRINIVAS, Romala Vijaya; PODILE, Venkateswararao; RADHIKA, T. Bankers' perception of big data for fraud detection and customer satisfaction. *AIP Conference Proceedings*. 2023, vol. 2821, no. 1, p. 040015. ISSN 0094-243X. Available from DOI: 10.1063/5.0158548.

78. FAWCETT, Tom. An introduction to ROC analysis. *Pattern Recognition Letters*. 2006, vol. 27, no. 8, pp. 861–874. ISSN 0167-8655. Available from DOI: 10.1016/j.patrec.2005.10.010. ROC Analysis in Pattern Recognition.

79. SAITO, Takaya; REHMSMEIER, Marc. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE*. 2015, vol. 10, no. 3, pp. 1–21. Available from DOI: 10.1371/journal.pone.0118432.

80. WIJENAYAKE, Senuri; HETTIACHCHI, Danula; HOSIO, Simo; KOSTAKOS, Vassilis; GONCALVES, Jorge. Effect of Conformity on Perceived Trustworthiness of News in Social Media. *IEEE Internet Computing*. 2021, vol. 25, no. 1, pp. 12–19. Available from DOI: 10.1109/MIC.2020.3032410.

81. JINDAL, Nitin; LIU, Bing. Opinion spam and analysis. In: *Proceedings of the 2008 International Conference on Web Search and Data Mining*. Palo Alto, California, USA: Association for Computing Machinery, 2008, pp. 219–230. WSDM '08. ISBN 9781595939272. Available from DOI: 10.1145/1341531.1341560.

82. OTT, Myle; CHOI, Yejin; CARDIE, Claire; HANCOCK, Jeffrey T. Finding deceptive opinion spam by any stretch of the imagination. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 309–319. HLT '11. ISBN 9781932432879.

83. BHAUMIK, Runa; WILLIAMS, Chad; MOBASHER, Bamshad; BURKE, Robin. Securing collaborative filtering against malicious attacks through anomaly detection. 2006.

84. ZHANG, Sheng; CHAKRABARTI, Amit; FORD, James; MAKEDON, Fillia. Attack Detection in Time Series for Recommender Systems. In: Philadelphia, PA, USA: Association for Computing Machinery, 2006, pp. 809–814. KDD '06. ISBN 1595933395. Available from DOI: 10.1145/1150402.1150508.

85. MUKHERJEE, Arjun; LIU, Bing; GLANCE, Natalie. Spotting fake reviewer groups in consumer reviews. In: *Proceedings of the 21st International Conference on World Wide Web*. Lyon, France: Association for Computing Machinery, 2012, pp. 191–200. WWW '12. ISBN 9781450312295. Available from DOI: 10.1145/2187836.2187863.

86. HSU, Chiao-Fang; KHABIRI, Elham; CAVERLEE, James. Ranking Comments on the Social Web. In: *2009 International Conference on Computational Science and Engineering*. 2009, vol. 4, pp. 90–97. Available from DOI: 10.1109/CSE.2009.109.

87. O'MAHONY, Michael P.; SMYTH, Barry. Learning to recommend helpful hotel reviews. In: *Proceedings of the Third ACM Conference on Recommender Systems*. New York, New York, USA: Association for Computing Machinery, 2009, pp. 305–308. RecSys '09. ISBN 9781605584355. Available from DOI: 10.1145/1639714.1639774.

88. WANG, Guan; XIE, Sihong; LIU, Bing; YU, Philip S. Review Graph Based Online Store Review Spammer Detection. In: *2011 IEEE 11th International Conference on Data Mining*. 2011, pp. 1242–1247. Available from DOI: 10.1109/ICDM.2011.124.

89. XIA, Hui; FANG, Bin; GAO, Min; MA, Hui; TANG, Yuanyan; WEN, Jing. A novel item anomaly detection approach against shilling attacks in collaborative recommendation systems using the dynamic time interval segmentation technique. *Information Sciences*. 2015, vol. 306, pp. 150–165. ISSN 0020-0255. Available from DOI: 10.1016/j.ins.2015.02.019.

90. GAO, Min; TIAN, Renli; WEN, Junhao; XIONG, Qingyu; LING, Bin; YANG, Linda. Item Anomaly Detection Based on Dynamic Partition for Time Series in Recommender Systems. *PLoS One*. 2015, vol. 10, no. 8. Available from DOI: 10.1371/journal.pone.0135155.

91. ZHOU, Wei; WEN, Junhao; GAO, Min; REN, Haijun; LI, Peng. Abnormal Profiles Detection Based on Time Series and Target Item Analysis for Recommender Systems. *Mathematical Problems in Engineering*. 2015, vol. 2015. Available from DOI: 10.1155/2015/490261.

92. ZHOU, Wei; WEN, Junhao; XIONG, Qingyu; ZENG, Jun; LIU, Ling; CAI, Haini; CHEN, Tian. Abnormal Group User Detection in Recommender Systems Using Multi-dimension Time Series. In: *Collaborate Computing: Networking, Applications and Worksharing*. Cham: Springer International Publishing, 2017, pp. 373–383. ISBN 978-3-319-59288-6.

93. YANG, Zhihai; CAI, Zhongmin; YANG, Yuan. Spotting anomalous ratings for rating systems by analyzing target users and items. *Neurocomputing*. 2017, vol. 240, pp. 25–46. ISSN 0925-2312. Available from DOI: 10.1016/j.neucom.2017.02.052.

94. YANG, Zhihai; SUN, Qindong; ZHANG, Yaling; ZHANG, Beibei. Uncovering anomalous rating behaviors for rating systems. *Neurocomputing*. 2018, vol. 308, pp. 205–226. ISSN 0925-2312. Available from DOI: 10.1016/j.neucom.2018.05.001.

95. MA, Xiaoxiao; WU, Jia; XUE, Shan; YANG, Jian; ZHOU, Chuan; SHENG, Quan Z.; XIONG, Hui; AKOGLU, Leman. A Comprehensive Survey on Graph Anomaly Detection With Deep Learning. *IEEE Transactions on Knowledge and Data Engineering*. 2023, vol. 35, no. 12, pp. 12012–12038. Available from DOI: 10.1109/TKDE.2021.3118815.

96. DOU, Yingtong; LIU, Zhiwei; SUN, Li; DENG, Yutong; PENG, Hao; YU, Philip S. Enhancing Graph Neural Network-based Fraud Detectors against Camouflaged Fraudsters. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 315–324. CIKM '20. ISBN 9781450368599. Available from DOI: 10.1145/3340531.3411903.

97. LIU, Yang; AO, Xiang; QIN, Zidi; CHI, Jianfeng; FENG, Jinghua; YANG, Hao; HE, Qing. Pick and Choose: A GNN-based Imbalanced Learning Approach for Fraud Detection. In: *Proceedings of the Web Conference 2021*. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 3168–3177. WWW '21. ISBN 9781450383127. Available from DOI: 10.1145/3442381.3449989.

98. PENG, Hao; ZHANG, Ruitong; DOU, Yingtong; YANG, Renyu; ZHANG, Jingyi; YU, Philip S. Reinforced Neighborhood Selection Guided Multi-Relational Graph Neural Networks. *ACM Trans. Inf. Syst.* 2021, vol. 40, no. 4. ISSN 1046-8188. Available from DOI: 10.1145/3490181.

99. ZENG, Yufan; TANG, Jiashan. RLC-GNN: An Improved Deep Architecture for Spatial-Based Graph Neural Network with Application to Fraud Detection. *Applied Sciences*. 2021, vol. 11, no. 12. ISSN 2076-3417. Available from DOI: 10.3390/app11125656.

100. WU, Bin; YAO, Xinyu; ZHANG, Boyan; CHAO, Kuo-Ming; LI, Yinsheng. SplitGNN: Spectral Graph Neural Network for Fraud Detection against Heterophily. In: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 2737–2746. CIKM '23. ISBN 9798400701245. Available from DOI: 10.1145/3583780.3615067.

101. XIANG, Sheng; ZHU, Mingzhi; CHENG, Dawei; LI, Enxia; ZHAO, Ruihui; OUYANG, Yi; CHEN, Ling; ZHENG, Yefeng. Semi-supervised credit card fraud detection via attribute-driven graph representation. In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press, 2023. AAAI '23. ISBN 978-1-57735-880-0. Available from DOI: 10.1609/aaai.v37i12.26702.

102. TAN, Xiaoyan; HENG, Yong; LI, Xin. Dual Channel Graph Neural Network for Fraud Detection. In: *Artificial Intelligence Logic and Applications*. Singapore: Springer Nature Singapore, 2023, pp. 241–254. ISBN 978-981-99-7869-4. Available from DOI: 10.1007/978-981-99-7869-4_19.

103. BROMLEY, Jane; GUYON, Isabelle; LECUN, Yann; SÄCKINGER, Eduard; SHAH, Roopak. Signature verification using a "Siamese" time delay neural network. In: *Proceedings of the 6th International Conference on Neural Information Processing Systems*. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1993, pp. 737–744. NIPS '93.

104. KOCH, Gregory; ZEMEL, Richard; SALAKHUTDINOV, Ruslan. Siamese Neural Networks for One-shot Image Recognition. In: *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France, 2015, vol. 37. JMLR W&CP. Available also from: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

105. AN, Na; QI YAN, Wei. Multitarget Tracking Using Siamese Neural Networks. 2021, vol. 17, no. 2s. ISSN 1551-6857. Available from DOI: 10.1145/3441656.

106. QIAO, Yuanyuan; WU, Yuewei; DUO, Fan; LIN, Wenhui; YANG, Jie. Siamese Neural Networks for User Identity Linkage Through Web Browsing. *IEEE Transactions on Neural Networks and Learning Systems*. 2020, vol. 31, no. 8, pp. 2741–2751. Available from DOI: 10.1109/TNNLS.2019.2929575.

107. SERRANO, Nicolás; BELLOGÍN, Alejandro. Siamese neural networks in recommendation. *Neural Computing and Applications*. 2023, vol. 35, no. 19, pp. 13941–13953. ISSN 1433-3058. Available from DOI: 10.1007/s00521-023-08610-0.

108. CHOPRA, Sumit; HADSELL, Raia; LECUN, Yann. Learning a similarity metric discriminatively, with application to face verification. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*. 2005, vol. 1, pp. 539–546. Available from DOI: 10.1109/CVPR.2005.202.

109. SCHROFF, Florian; KALENICHENKO, Dmitry; PHILBIN, James. FaceNet: A unified embedding for face recognition and clustering. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 815–823. Available from DOI: 10.1109/CVPR.2015.7298682.

110. ZHOU, Xiaokang; LIANG, Wei; SHIMIZU, Shohei; MA, Jianhua; JIN, Qun. Siamese Neural Network Based Few-Shot Learning for Anomaly Detection in Industrial Cyber-Physical Systems. *IEEE Transactions on Industrial Informatics*. 2021, vol. 17, no. 8, pp. 5790–5798. Available from DOI: 10.1109/TII.2020.3047675.

111. PANG, Guansong; SHEN, Chunhua; JIN, Huidong; HENGEL, Anton van den. Deep Weakly-Supervised Anomaly Detection. In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Long Beach, CA, USA: Association for Computing Machinery,

2023, pp. 1795–1807. KDD '23. ISBN 9798400701030. Available from DOI: 10.1145/3580305.3599302.

112. YUAN, Tongtong; DENG, Weihong; TANG, Jian; TANG, Yinan; CHEN, Binghui. Signal-To-Noise Ratio: A Robust Distance Metric for Deep Metric Learning. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA, 2019, pp. 4810–4819. Available from DOI: 10.1109/CVPR.2019.00495.

113. HU, Ziniu; DONG, Yuxiao; WANG, Kuansan; SUN, Yizhou. *Heterogeneous Graph Transformer*. 2020. Available from DOI: 10.48550/arXiv.2003.01332.

114. ZHANG, Chuxu; SONG, Dongjin; HUANG, Chao; SWAMI, Ananthram; CHAWLA, Nitesh V. Heterogeneous Graph Neural Network. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 793–803. KDD '19. ISBN 9781450362016. Available from DOI: 10.1145/3292500.3330961.

115. WANG, Xiao; JI, Houye; SHI, Chuan; WANG, Bai; CUI, Peng; YU, P.; YE, Yanfang. *Heterogeneous Graph Attention Network*. 2021. Available from DOI: 10.48550/arXiv.1903.07293.

116. MANN, H. B.; WHITNEY, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*. 1947, vol. 18, no. 1, pp. 50–60. Available from DOI: 10.1214/aoms/1177730491.

117. JR., Frank J. Massey. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*. 1951, vol. 46, no. 253, pp. 68–78. Available from DOI: 10.1080/01621459.1951.10500769.

118. HUANG, Qiang; YAMADA, Makoto; TIAN, Yuan; SINGH, Dinesh; YIN, Dawei; CHANG, Yi. *GraphLIME: Local Interpretable Model Explanations for Graph Neural Networks*. 2020. Available from DOI: 10.48550/arXiv.2001.06216.

119. RAYANA, Shebuti; AKOGLU, Leman. Collective Opinion Spam Detection: Bridging Review Networks and Metadata. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 985–994. KDD '15. ISBN 9781450336642. Available from DOI: 10.1145/2783258.2783370.

120. ZHANG, Shijie; YIN, Hongzhi; CHEN, Tong; HUNG, Quoc Viet Nguyen; HUANG, Zi; CUI, Lizhen. *GCN-Based User Representation Learning for Unifying Robust Recommendation and Fraudster Detection*. 2020. Available from DOI: 10.48550/arXiv.2005.10150.

121. ROZEMBERCZKI, Benedek; ALLEN, Carl; SARKAR, Rik. *Multi-scale Attributed Node Embedding*. 2021. Available from DOI: 10.48550/arXiv.1909.13021.

122. HEIKE HOFMANN, Hadley Wickham; KAFADAR, Karen. Letter-Value Plots: Boxplots for Large Data. *Journal of Computational and Graphical Statistics*. 2017, vol. 26, no. 3, pp. 469–477. Available from DOI: 10.1080/10618600.2017.1305277.

Contents of Attachments

- `README.MD` ..... description of the files
- `text/` ..... directory containing the thesis and LaTeX source files
  - `bohundan-assignment.pdf` ..... thesis assignment
  - `bohundan-thesis.tex` ..... source of this thesis
  - `FITthesis.cls` ..... modified document template
  - `Makefile` ..... Makefile for compiling this thesis
  - `references.bib` ..... BibTeX references
- `media/` ..... directory containing all figures and graphics
  - `figures/` ..... directory containing figure drawings
  - `graphs/` ..... directory containing experiment graphs
- `src/` ..... directory containing the implementation of the proposed model
  - `config.py` ..... hyperparameter configuration file
  - `datasets.py` ..... dataset loaders
  - `distances.py` ..... distance functions
  - `losses.py` ..... loss functions
  - `main_coupled.py` ..... training script for coupled version
  - `main_decoupled.py` ..... training script for decoupled version
  - `main_mlp.py` ..... training script for multi-layer perceptron
  - `models.py` ..... model implementations
  - `operators.py` ..... heterogeneous graph convolution operators
  - `requirements.txt` ..... Python package version list
  - `samplers.py` ..... implementation of data samplers
  - `utils.py` ..... evaluation utilities
- `datasets/` ..... directory containing processed dataset files
  - `Amazon.mat` ..... Amazon dataset
  - `FDCompCN.mat` ..... FDCompCN dataset
  - `S-FFSD.mat` ..... S-FFSD dataset
  - `Yelp.mat` ..... Yelp dataset