UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

Recommendation Systems based on Graph Neural Networks architectures: MovieLens case study

Master's Thesis

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

AUTHOR: Beltrán Domínguez, Victoria

Tutor: Onaindia de la Rivaherrera, Eva

ACADEMIC YEAR: 2021/2022

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Recommendation Systems based on Graph Neural Networks architectures: MovieLens case study

TRABAJO FIN DE MASTER

Master Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Author: Victoria Beltrán Domínguez
Tutor: Eva Onaindía de la Rivaherrera

Course 2021-2022

Resum

Els sistemes de recomanació ajuden als usuaris a trobar elements rellevants i adequats entre l'enorme quantitat d'opcions disponibles. En aquest projecte, explorem de quina manera es pot construir un sistema de recomanació utilitzant Xarxes Neuronals basades en Grafs (GNNs), un nou tipus de xarxes neuronals que operen directament en l'estructura nativa de les dades. En particular, utilitzant el conjunt de dades de MovieLens, compararem dos enfocaments diferents. El primer enfocament es basa en un graf bipartit on els nodes representen usuaris i pel·lícules, i les arestes indiquen les relacions entre estos i, l'altre enfocament es centra en el Processament de Senyals de Grafs. En ambdós enfocaments, realitzem diferents experiments a fi de comparar els resultats. Tot el codi desenvolupat en aquest projecte està escrit en PyTorch Geometric, una biblioteca basada en PyTorch que facilita la manipulació de les GNNs.

Paraules clau: Sistemes de Recomanació, Xarxes Neuronals de Grafs, Machine Learning, Processament de Senyals de Grafs

Resumen

Los sistemas de recomendación ayudan a los usuarios a encontrar elementos relevantes y adecuados entre la enorme cantidad de opciones disponibles. En este proyecto, exploramos de qué manera se puede construir un sistema de recomendación utilizando Redes Neuronales basadas en Grafos (GNNs), un nuevo tipo de redes neuronales que operan directamente en la estructura nativa de los datos. En particular, utilizando el conjunto de datos de MovieLens, compararemos dos enfoques diferentes. El primer enfoque se basa en un grafo bipartito donde los nodos representan usuarios y películas, y las aristas indican las relaciones entre estos y, el otro enfoque se centra en el Procesamiento de Señales de Grafos. En ambos enfoques, realizamos diferentes experimentos con el fin de comparar los resultados. Todo el código desarrollado en este proyecto está escrito en PyTorch Geometric, una biblioteca basada en PyTorch que facilita la manipulación de las GNNs.

Palabras clave: Sistemas de Recomendación, Redes Neuronales de Grafos, Machine Learning, Procesamiento de Señales de Grafos

Abstract

Recommendation systems help users find relevant and suitable items among the enormous amount of selectable choices. In this project, we explore how a recommendation system can be built using Graph Neural Networks (GNNs), a new type of neural network that operates directly on the native graph structure of the data. Particularly, using the MovieLens dataset, we will compare two different approaches. The first approach is based on a bipartite graph where nodes represent users and movies, and the edges denote relationships among them. The other approach is centered on Graph Signal Processing. In both approaches, different experiments will be carried out in order to compare the results. All the code developed in this project is written in PyTorch Geometric, a library built upon PyTorch that eases the manipulation of GNNs.

Key words: Recommendation Systems, Graph Neural Networks, Machine Learning, Graph Signal Processing

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

Nowadays, we have an overwhelming offer of products and services. Google, Amazon, Netflix(source), Instagram, Facebook, YouTube [2], Tinder ... the vast majority of platforms have integrated recommendation systems that guide and advise their users. And they represent big advantages both for users and companies (source).

The existence of a recommendation system represents an advantage as they allow users to discover products of interest in the shortest possible time. In addition, it serves to promote new products that may be of interest as well as to remember when they are on sale.

Furthermore, companies are also critically impacted by this type of systems since the income is usually boosted with the use of an appropriate recommendation system. Users find in shortest time what they need, and in exchange, a large number of purchases is accomplished [3].

That being the case, there is no doubt about the importance of the role of recommender systems in web platforms and services. The vital information to be able to create a recommendation system is usually represented as a series of relationships or interactions (buying, clicking, rating, adding it to a wish list, etc.) of a user with a product. Moreover, we can also take into account particular characteristics of both products and users to make a very fine-grained recommendation. However, this information is not easy to represent nor manipulate in the strict way of machine learning, being the easiest way to represent both the characteristics of user, items and its relationships as a graph.

In this way, we will explore how a recommendation system can be built using Graph Neural Networks (GNNs), a new type of neural network that operates directly on the native graph structure of the data.

For this case, using the MovieLens dataset [4], two different approaches will be explained and compared in order to find out if this type of system really manages to take advantage of all the available information and its relationships.

1.1 Motivation

This project arises as an initiative for a VRAIN research project to evaluate GNNs as a possible tool for a recommender system. In particular, the objective of the project is to create a recommendation system in order to encourage sustainable tourism around the community of Valencia. Unfortunately, due to certain complications, we have not been able to train a model with the expected data.

Nevertheless, on a personal level, the idea of working with GNNs, which have a different approach of what I have learnt throughout the course, was very interesting to me. They are innovative and dynamic networks, that consume very few resources and help process graph-like data. On the other hand, at a professional level I consider that they are not simply interesting, but that they have potential, since despite their novelty, they have already achieved results comparable to the state of the art within the field of recommendation with relatively simple architectures. Several GNN applications have obtained excellent results:

- Decathlon Canada, a general sporting goods retailer with online shopping capabilities, conducted an experiment to contrast GNNs with conventional models (matrix factorization, RNN, nearest neighbours, etc). Results show that the most popular items baseline is considerably outperformed by the GNN model. Decathlon subject-matter experts claim that the GNN model produces outcomes that are comparable to those of the already-in-use models. Additionally, GNN approach employs representational learning, and therefore, the learnt embeddings may be helpful for other tasks.

- Another work proposes a Graph Neural Networks in order to predict user actions based on anonymous sessions [5]. In the proposed method, information about the session sequences is modeled as graph. Based on this session graph, GNN can capture complex transitions of items, which are difficult to be revealed by previous conventional sequential methods. Each session is then represented as the composition of the global preference and the current interest of that session using an attention network. Extensive experiments conducted on two real datasets show that SR-GNN evidently outperforms the state-of-the-art session-based recommendation methods consistently.

- The results of some investigations showed to outperform state of the art results, by proposing a novel Graph Neural Network that uses different node types as well as rich node attributes to model data in the recommendation system [6]. In this proposal, they design a component connecting potential neighbors to explore the influence among neighbors and provide two different strategies with the attention mechanism to aggregate neighbors' information.

Given the interest on this type of network and the inconvenience of the original dataset, MovieLens has served as the dataset to experiment and evaluate the results.

1.2 Objectives

In this section, we present the objectives for this project:

- Be able to interpret the underlying theoretical concepts of Graph Neural Networks.

- Create two recommendation system using two specific different types of GNN.

- Evaluate obtained results using MovieLens dataset.

- Compare both approaches results.

1.3 Memory Structure

This section proceeds to present an annotated index of the memory structure:

- **Chapter 1: Introduction**

  This chapter introduces the main subject of this project, as well as the motivation that has propelled the realization of the project and the objectives pursued with it. The structure of the memory that will be followed throughout is also defined.

- **Chapter 2: Background**

  In this third chapter, the basic concepts and principal notions of Graph Neural Networks are explained in order for the reader to understand the contribution of this project.

- **Chapter 3: Recommendation with Graph Signal Processing**

  Fourth chapter exposes the model build using GSP, describing the machine learning pipeline and some implementation details. A series of experiments are also presented, with their respective results in order to evaluate the model and its effectiveness.

- **Chapter 4: Recommendation with Bipartite Graph**

  Fifth chapter exposes the model build using structuring available information as a bipartite graph, describing the machine learning pipeline and some implementation details. A series of experiments are also presented, with their respective results in order to evaluate the model and its effectiveness.

- **Chapter 5: Conclusions and future work**

  In this last chapter, we present the conclusions of this project, comparing both approaches and their respective results and highlighting the objectives that have been achieved. Finally, possible future lines of work are presented.

CHAPTER 2

Background

This chapter is intended to provide the reader the principal concepts and notions on Graph Neural Networks (GNNs) with the aim to understand the contribution of this project, which are presented in the following two chapters.

This chapter is structured as follows; section 2.1 presents an overview on GNNs; section 2.2 introduces basic concepts of *Spectral GNNs* and *Graph Signal Processing*; section 2.3 introduces the basic theory of *Spatial based Graph Neural Networks*; section 2.4 explains how to adapt both approaches in order to create a recommendation system; and section 2.5 presents the dataset we will use in our experimentation.

2.1 Overview of Graph Neural Networks

A *Graph Neural Network* (GNN) is a type of neural network which directly operates on the graph structure and helps resolve problems where data is naturally presented as a graph. This advantage alleviates the loss of information on the relationships among data when using neural networks.

A graph $\mathcal{G}$ is a structure made of a set of vertices $\mathcal{V}$, a set of edges $\mathcal{E}$ between those vertices, and a set of weights $\mathcal{W}$ for each edge. Specifically:

- Vertices or nodes are a set of n labels from $\{1, ..., n\}$.
- Edges are ordered pairs of labels *(i,j)*, where each label represents a node. An edge *(i,j)* means that node i **has a connection to node** j **or that** i **can be influenced by** j.
- Weights $w_{ij} \in \mathbb{R}$ represent the **strength of the influence of node** j **on node** i. If no weights are specified on a graph, all edges have the same relevance or influence.

Graphs can be directed or undirected, and weighted or unweighted. We will mostly work with undirected graphs (both weighted and unweighted versions). An exemplification of the two types of graphs can be found in Figures 2.1 and 2.2. Undirected edges are represented as a straight line with no arrows between two nodes or, as shown in the Figures, as a bidirectional arrow between two nodes ($\leftrightarrow$). Additionally, GNNs work over graphs that have features associated to the nodes and/or to the edges.

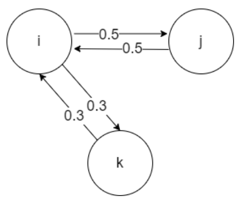

Figure 2.1: Undirected weighted graph

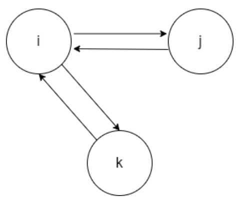

Figure 2.2: Undirected unweighted graph

Several different types of tasks can be solved with GNNs [7]: node-level tasks (node regression and node classification), edge-level tasks (edge classification and link prediction) and graph-level tasks (graph classification). Although semi-supervised and unsupervised learning is possible, in this project we will work with supervised learning.

There exists different types of GNNs. One possible categorization based on the architecture separates GNNs into *Recurrent Graph Neural Networks*, *Graph Convolutional Networks*, *Graph Autoencoders* and *Spatial-Temporal Graph Neural Networks* [7]. Amongst all the mentioned types of GNN, we will be focusing on *Graph Convolutional Networks* (GCN), a type of GNN that generalizes the *convolution* operation from grid data to graph data. The main idea of a GCNs is to generate the representation of a node by aggregating its own features with its neighbors features. GCNs fall into two categories: spectral-based approaches and spatial-based approaches.

In this project, we will work with *Spectral GCNs* and *Spatial GCNs*. Both approaches can be implemented as a message passing network. The intuition behind GCNs is that nodes are naturally defined by their neighbors and connections. Therefore, several common steps are performed in these networks:

1. **Message Passing**: Message passing networks were introduced in [8] and constitute the basis of the information flow in GNNs. For simplicity, let us assume that node i has an initial feature vector h_i^0. A feature vector associated to a node represents the features of that node. However, though message passing, our aim will be to learn an embedding h_i^k for each node, which encodes information of the node itself in a low-dimensional space. Even though both will be represented by the letter h, the feature vector h_i^0 is only associated with time 0, as it represents the initial information of the node itself. In order to learn embeddings, information is propagated a fixed number of times. This information is propagated following three steps:

    - **Transform**: Each node i computes the message for its neighbors. Messages are functions that help summarize the node's information. Once the message is prepared for each neighbour, they are all sent simultaneously.
    - **Aggregate**: At each node, neighbor's messages arrive. This step aims to apply a function that aggregates all the received messages, using a permutation-invariant function (usually average or sum).
    - **Update**: combining the aggregated final message of the neighbours and the current embedding of the node, it updates itself.

    The previously explained parts can be mathematically viewed as:

    $$h_i^{(k+1)} = UPDATE(h_i^k, AGGREGATE_{j \in N(i)} TRANSFORM(h_j^k)) \tag{2.1}$$

    This process occurs synchronously for every node in the graph, updating nodes at every message passing step k. Each step is called a graph convolutional layer. Several layers can be applied sequentially.

2. **Readout**: Once several layers have been applied, a fully-connected layer or MLP is used to predict our target.

In the following sections we will focus on the two approaches that will be used in this project: *Spectral GCN* based on *Graph Signal Processing* and *Spatial-based GCNs*.

2.2 Spectral GCN: Graph Signal Processing

Spectral Graph Convolutional Networks are a type of GCN that facilitate the understanding of the underlying structure of the graph by identifying clusters or sub-groups in the spectral domain. Intuitively, these networks help predict how the information will propagate over a graph. The methods that define the convolution operator in spectral GCN are based on the *Graph Signal Processing* theory.

The contents of this section are mostly taken from [9], a course that explains in a simple and very transparent way how to use GNNs combined with Graph Signal Processing for recommendation. In fact, most of the papers found on this field are basically from the same research group and it appears that this is still an ongoing research line.

Graph Signal Processing is a research field that deals with signals whose domain is not ordered along some axis but on a graph. Therefore, signal processing applied on graphs studies data over irregular structures. Graph Signal Processing provides the necessary tools to analyze the effects of a changing property or attribute of an object which is related somehow to other objects of the same type or different type.

A graph signal is a vector $\mathrm{x} \in \mathbb{R}^n$ in which component x_i is associated with node i. Graph signals are the objects to be processed when information is propagated.

Figure 2.3 shows a five-node undirected and weighted graph $\mathcal{G}$ ($n = 5$). The graph $\mathcal{G}$ represents an expectation of proximity or similarity between components. The weight w_{12} of the edge between node 1 and node 2 denotes the similarity between these two nodes. Similarly, the absence of an edge represents the lack of similarity between the nodes. For example, in $\mathcal{G}$, node 2 and node 4 are not found to be similar in any way.

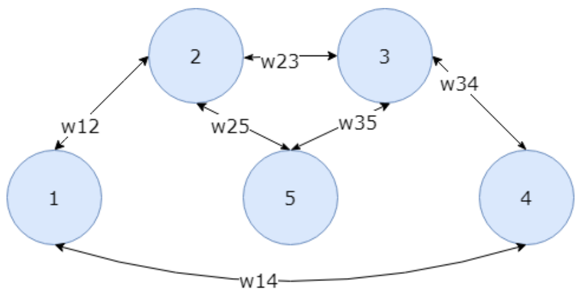

Figure 2.3: Graph $\mathcal{G}$

Let's suppose that the nodes in the graph $\mathcal{G}$ represent movies and weights on edges express the similarity between movies; and a user has seen some of the movies, rating them from 1 to 5. Specifically:

- Movie 1: 3 stars
- Movie 2: 4 stars
- Movie 3: Not seen.

- Movie 4: Not seen.
- Movie 5: 1 star.

The user's preferences would be represented as a graph signal vector of 5 components, where each position contains a scalar associated to the corresponding node (movie) in the graph: $[3, 4, 0, 0, 1]$. This graph signal is represented in Figure 2.4.

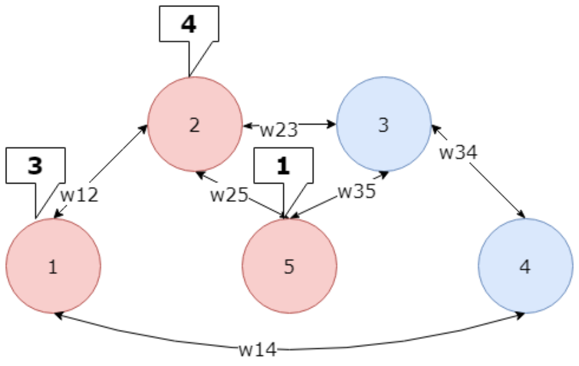

Figure 2.4: Graph signal representation in graph $\mathcal{G}$

In this example, a graph signal represents the preferences of a user, so we would have as many graph signals as users rating the different movies.

2.2.1. Graph Shift Operator

Given that the native structure of our target data is represented via a graph, we need a mechanism that allows us to represent and work with the graph.

In the context of Graph Signal Processing, the Graph Shift Operator (also known as GSO) is a matrix representation of the graph. The GSO is intrinsic to the graph signal as it represents similarities between the nodes indexed in the signals. Specifically, the GSO acts as a support for the data comprised in the graph signal describing the arbitrary pairwise relationships between the data elements. Standard GSO matrices are the Adjacency and Laplacian matrices, as well as the respective normalized versions.

- **Adjacency Matrix** A: Sparse matrix A defined as:

$$A_{ij} = w_{ij} \;\; \forall (i,j) \in \mathcal{E} \tag{2.2}$$

- **Laplacian Matrix** L: Sparse matrix L defined as:

$$L = D - A \tag{2.3}$$

where D represents the degree matrix. The degree matrix is a diagonal matrix where each entry is defined as:

$$D(i,j) = \left\{ \begin{array}{ll} deg(v_i), & \text{if i = j} \\ 0, & \text{otherwise} \end{array} \right\} \tag{2.4}$$

where the degree $deg(v_i)$ of a vertex sums the weights of the incidents edges.

The Laplacian Matrix L measures to what extent a vertex in the graph differs from the values of the nearby vertices.

Since matrices A and L are used to diffuse graph signal components across their respective neighbours, they will typically contain large values that could eventually end up in the so-called exploding gradient problem, where optimization is very difficult. To solve this problem, matrix normalization is used, which consists in multiplying the target matrix before and after by the degree matrix $D^{-\frac{1}{2}}$. Adopting the normalized versions of the GSO help overcome the difficulties that arise by the vanishing and exploding gradient problem.

- **Adjacency Normalized Matrix $\bar{\mathbf{A}}$**:

$$\bar{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{2.5}$$

- **Laplacian Normalized Matrix $\bar{\mathbf{L}}$**:

$$\bar{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \tag{2.6}$$

Shortly, this normalization averages the neighbours of a node while taking into account the number of neighbours of its neighbours. This means that matrix normalization will weigh the importance of a connection: if a node is connected to a node that is highly connected, the connection is less relevant than if it is linked to a node with fewer connections. Mathematically, given our GSO and its respective degree matrix D, the operation $D^{-\frac{1}{2}}(GSO)D^{-\frac{1}{2}}$ effectively accomplishes the previously explained normalization. Once the GSO is normalized, its values will be in a similar range, which will aid to alleviate the vanishing and exploding gradient problem.

The specific choice of GSO matters in practice but most of the results and analysis hold for any type of GSO.

2.2.2. Graph Signal Diffusion

In this section we will explain how information is propagated across the nodes of the graph, providing both an intuitive explanation and a mathematical foundation.

The aim is to build a model that, given a graph signal, predicts the values for the non-observed (unknown) components of the signal. For example, if the nodes of a graph denote movies, and a graph signal represents the ratings given by a user to those movies, the objective will be to predict the ratings of the movies the user has not seen by taking into account the movies that the user did see.

In order to accomplish this behaviour, we need a way to propagate information between nodes using the similarity value between them. This is done through a mechanism called *diffusion*. Diffusion consists in multiplying the Graph Shift Operator (we will denote the GSO with the letter S hereafter) by a graph signal. This operation accomplishes the diffusion of a graph signal over the graph. Therefore, the diffused signal y is expressed as $y = Sx$; for a particular component i of the signal, given that $n(i)$ are the neighboring components associated, we have:

$$y_i = \sum_{j \in n(i)} w_{ij} x_j \tag{2.7}$$

Specifically, the components of a graph signal are only affected by the components which are marked to be similar in the graph. Figure 2.5 shows a situation where the focus is on the second position of the signal (node 2). This position will only be affected by the neighbours marked in green (that is 1, 3 and 5), whereas 4 will not have any influence on its diffused value. Obviously, stronger weights will contribute more to the diffusion output.

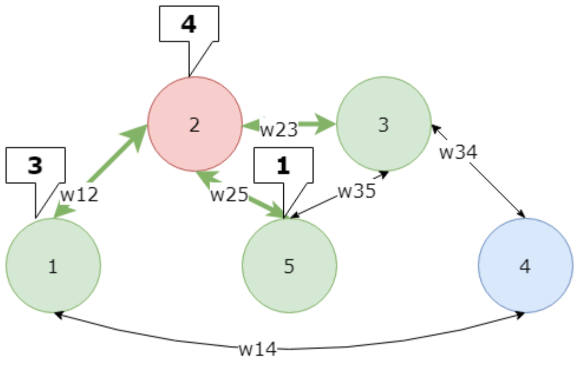

Figure 2.5: Nodes involved in the diffusion of node 2

Applying the diffusion operation to an already diffused graph signal produces a diffusion sequence. Basically, this can be viewed as applying k times the diffusion operation over the same graph signal. Mathematically, this process is equivalent to:

$$x^{(k+1)} = Sx^k \text{ with } x^{(0)} = x \tag{2.8}$$

We must remark that diffusion is a local operation wherein each graph signal component is combined with the components associated to its neighboring nodes. Thus, a diffusion sequence $x^{(k)}$ diffuses information to and from k-hop neighbourhoods. For low values of k, the entries of the diffusion sequence represent local information only, whereas, for large values of k, the entries of the diffusion sequence represent global information. This trade off between local and global information is beneficial and will be used in the definition of graph convolutions.

2.2.3. Graph Convolutional Filters

In this section we will explain how the diffusion mechanism is used to create a Graph Convolutional Filter.

Let us start with the definition of a filter. A filter H is a vector of K components $[h_0, h_1...., h_{K-1}]$ where h_i is known as a *filter tap* and K is the number of diffusion operations that are applied. A Graph Convolutional Filter is an operator that, given a graph signal, applies a diffusion sequence to such signal, weighing each propagation with the corresponding filter tap of H. Mathematically, it can be viewed as:

$$y = \sum_{k=0}^{K-1} h_k S^k x \tag{2.9}$$

Intuitively, the graph signal x is propagated K times, giving different relevance to each and every propagation. By doing so, depending on the particular characteristics of the task, the filter can help discern more critical diffusion steps from the ones that may not handle that much information. The first diffusion step may not have the same relevance as the further ones, depending on which information is more relevant for the task (local or global information).

Let us exemplify how the information is propagated, making a trade off between local and global information, by looking at the example of Figure 2.6. We will exclusively focus on a particular node, node x_4, but the same process is applied to all the nodes.

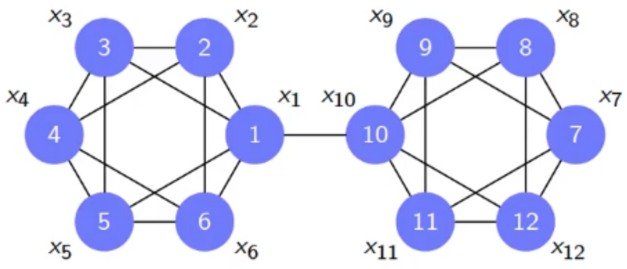

Figure 2.6: Reference Graph

As can be seen in Figure 2.7, we begin with the value of the node itself which is the associated component of the graph signal x_4.

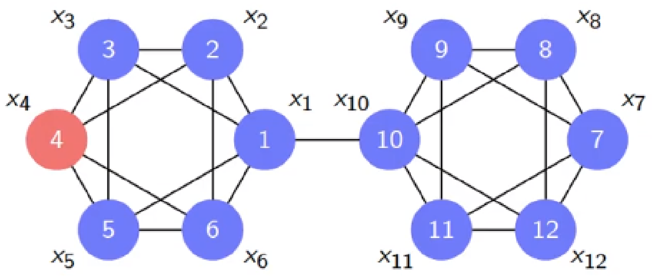

Figure 2.7: $y = h_0 S^0 x = x$

It must be noticed that the formulas exposed in Figure 2.7 are representing the shifted graph signal, where the component y_4 will be a value transformed by the average of the GSO application over x_3 x_2 x_5 and x_6. Therefore, in order to add information about one hop neighbours, we add to this graph signal the diffusion Sx multiplied by h_1 (following Equation 2.9). The convolution output on x_4 becomes a diffusion sequence affected by all of its one-hop neighbours (the ones marked in red in Figure 2.8). Mathematically, after a one-hop neighbour diffusion, the diffused signal y will be equal to $h_0 S^0 x + h_1 S^1 x$.

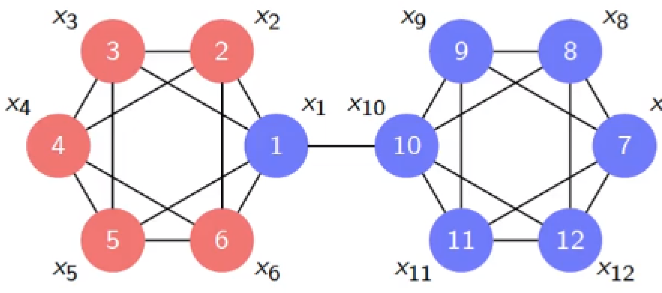

Figure 2.8: $y = h_0 S^0 x + h_1 S^1 x$

For taking into account information about two hop neighbors, again we add again $h_2 S^2 x$ (as can be seen in Figure 2.9). Mathematically, graph signal x propagated through its two hop neighbors would be equal to $y = h_0 S^0 x + h_1 S^1 x + h_2 S^2 x$.

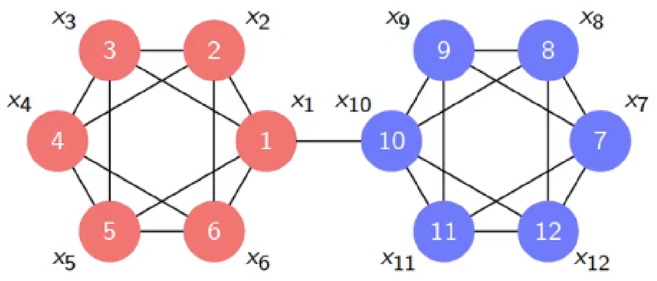

Figure 2.9: $y = h_0 S^0 x + h_1 S^1 x + h_2 S^2 x$

Summarizing, a graph convolutional filter is a weighted linear combination of the elements of the diffusion sequence. Similarly to image convolutions, depending on the number of convolutions applied, local or global information will be considered. Our aim will be to learn the most appropriate filters in order to weigh which convolutions and neighbors are the most important [10].

We can extend the descriptive power of graph signals by assigning an F-dimensional vector to each node instead of a simple scalar, resulting in feature vectors associated for each node. As we can notice, the diffusion mechanism is actually doing the same process as the message passing phase, whereas the linear layers will be added at the end of the process in order to compose the MLP for readout phase.

2.3 Spatial-based GCN

As we have previously explained, *Graph Convolutional Networks* can be divided into spectral-based approaches and spatial-based approaches. In this section we address the second type of GCN.

Spatial-based GCNs define graph convolutions based on the spatial relations between nodes. In other words, the feature vector or embedding of a node is given a geometric interpretation (spatial position) and the distance between nodes is leveraged for the task of link prediction. Link prediction is essential in social networks to infer social interactions or to suggest possible friends to the users, and it is also used in recommendation system and in predicting criminal associations.

For this task, the goal is to train a GNN that is capable to predict not only if a link exists between two given nodes, but to quantify its relationship. This is done by using the information of the node feature vectors as well as the edge features.

Given a set of attributes or features, GNNs use Message Passing to gather and learn the node embeddings. Similarity scores of node embeddings are used to determine whether or not two nodes should be connected. The weights of the neural network will be tuned through a loss function computed on predictions during the training phase [11]. Following we present an example to show the functioning of this procedure.

Let us consider the graph shown in Figure 2.10. Nodes represent elements of the problem with associated features. In our particular example, nodes denote people and features represent aspects associated with the person (gender, age, job, etc). It is important that all nodes of the same type have the same information represented in their feature vectors, as they will be used to compute the similarity between them.

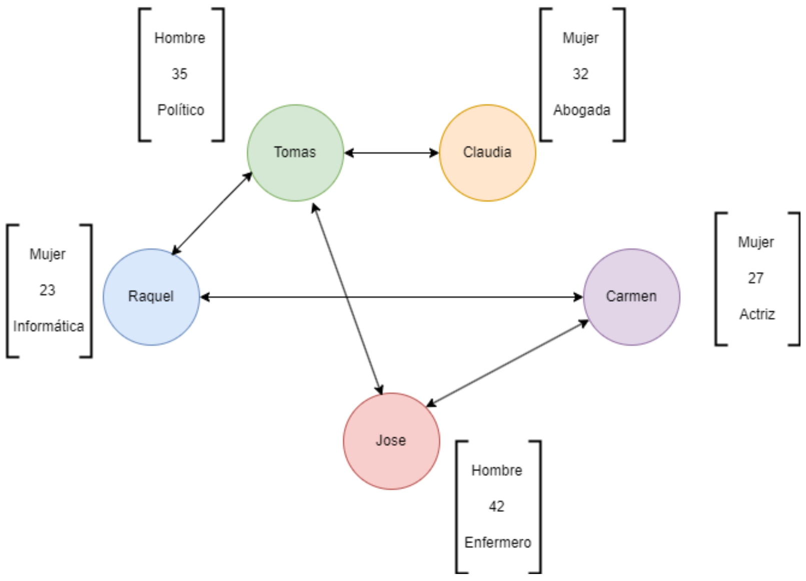

Figure 2.10: Social network

With this information, we want to predict whether two nodes will be connected, particularly whether there should be a link between Tomas and Carmen. To this end, we need to apply Message Passing. Let us put the focus on Tomas. The current neighbours of node Tomas (*Raquel*, *Jose* and *Claudia*) prepare and send a message. The information from all the received messages is summarized (sum, average, etc); then, according to some update function that takes into account both the feature vector of Tomas and the summarized info, a new feature embedding is assigned to Tomas. This process is shown in Figure 2.11.

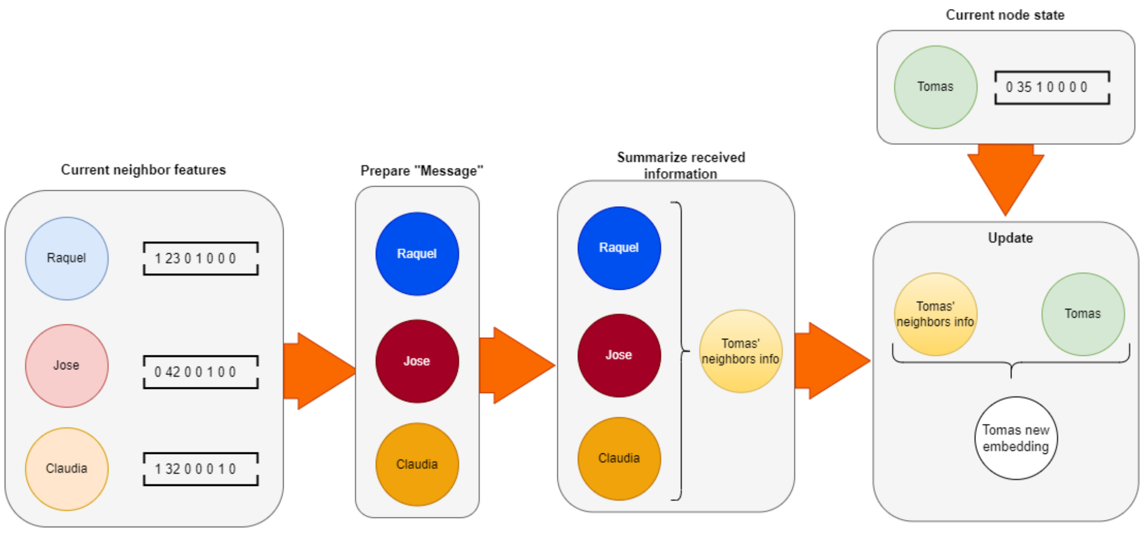

Figure 2.11: Message Passing for node Tomas

This process is mathematically expressed in equation 2.10 where h_i^t is the embedding associated with node i at time t, W and U are trainable parameters that ponderate the importance of both the node itself and its neighbors respectively, and $N(i)$ are the neighbors of i:

$$h_i^{k+1} = \sigma(Wh_i^k + U \sum_{j \in N(i)} \frac{h_j^t}{|N(i)|}) \tag{2.10}$$

The message passing occurs for every node simultaneously K times or layers. After that, h_i^K is the final embedding of node i. Finally, to train the model we need a readout layer that generates the predictions. Then, a loss function will be applied, back propagating the error in order to train the weight parameters W and U. The formula presented is a simple general convolution, but specific convolutions will be defined in future sections.

2.4 GNNs in recommendation

In this section we will explore the application of GNNs to address the problem of recommending items to a user.

Recommendation Systems (RSs) help users find desired items using the product similarities and closeness of the user preferences. Three main approaches are considered in RSs: **content-based approach** (where recommendation is based on similar items with similar features); **collaborative filtering approach** (where recommendation is based on items rated by similar users); and **hybrid approaches** that combine content-based and collaborative filtering techniques. Typically, collaborative filtering requires fewer assumptions than content filtering and yields a superior performance in real datasets. There exists two main techniques to implement a collaborative filtering algorithm: Latent Linear Factor (Matrix Decomposition) and the Nearest Neighbour algorithm [12].

2.4.1. Recommendation with Graph Signal Processing

In this project, we focus on the implementation of the Nearest Neighbour approach using Graph Signal Processing. Let us assume we have:

- U users indexed by u.
- I items indexed by i.
- A rating matrix X where X_{ui} represents the rating that user u **has given** to item i.
- A rating matrix $\overline{X}$ where $\overline{X}_{ui}$ represents the rating that user u **would give** to item i (unobserved). Notice that while matrix X represent already stated ratings by the user, matrix $\overline{X}$ represent hypothetical ratings if user had rated all the items.
- A rating matrix $\hat{X}$ where $\hat{X}_{ui}$ represents the **predicted rating** of the user u on item i.
- Ratings by the u-th user: $x_u = [X_{u,1}, ..., X_{u,I}]$
- Ratings to the i-th item: $x^i = [X_{1,i}, ..., X_{U,i}]$

Our problem, using Collaborative Filtering, will be that given the observed ratings in the matrix X, we should predict $\hat{X}$, as similar as possible to the real unobserved matrix $\overline{X}$ [12].

In order to illustrate this problem, let us introduce Figure 2.12, where several matrices are shown. In these matrices, rows represent users, columns represent items and colors represent different type of ratings assigned from users to items. The first matrix, $\overline{X}$, represents all the ratings that the users would give to all of the items in an hypothetical world. Second matrix, X, represents observed ratings, the available information that we have. Through the exploitation of similarities of users, and given a graph signal x_u, our aim will be to predict matrix $\hat{X}$ as similar as possible to $\overline{X}$.

In other words, our aim is to estimate predictions of unobserved ratings exploiting the similarities between users or between items.

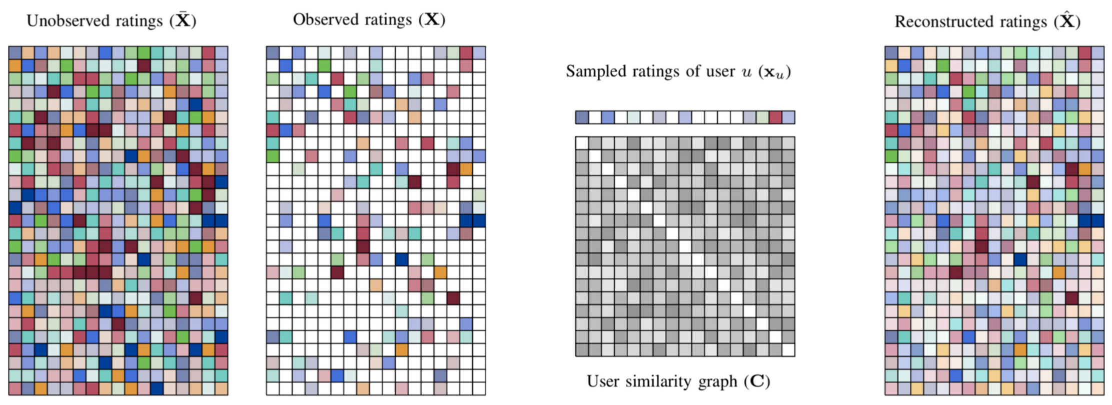
Unobserved ratings ($\overline{\mathbf{X}}$)
Observed ratings ($\mathbf{X}$)
Sampled ratings of user u ($\mathbf{x}_u$)
User similarity graph ($\mathbf{C}$)
Reconstructed ratings ($\hat{\mathbf{X}}$)

Figure 2.12: *Problem 1* graphically represented

To this end, we need first to decide if our graph embodies similarities between users or items.

- **User-based approach**: In this approach, nodes in the graph represent users, edges represent similarities between users, and the graph signals are the ratings given to one item x^i by all the users.

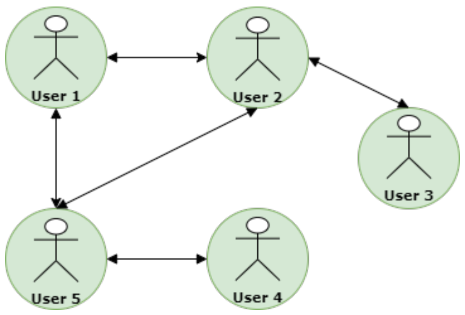

Figure 2.13: Graph representing similarities between users.

- **Item-based approach**: In this approach, nodes in the graph represent items, edges represent similarities between items, and the graph signals are the ratings given by one user x_u to all the items.

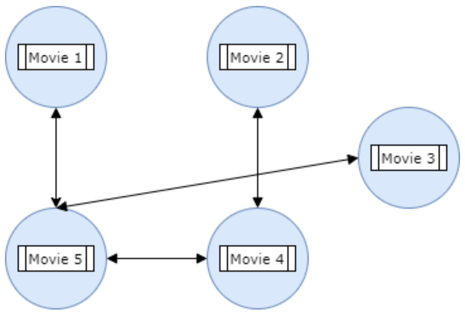

Figure 2.14: Graph representing similarities between items.

For a given graph signal x (user or item), applying a graph convolutional filter will result in a linear transformation that propagates information through the neighbours. This way, most similar users or items will be used to guide when making predictions, just like in the classical Nearest Neighbour approach.

The graph convolutional filters can be seen as typical image convolutions. Intrinsically, the same is done in both approaches, combining neighborhood information and weights in order to make the best predictions. However, images usually have predefined neighbours, as in a grid. For this reason, GCN make a neighboring structure and adapt it to work practically the same as CNN [10].

2.4.2. Recommendation with Bipartite Graph Neural Networks

In this section we present how to apply the *Spatial based Graph Neural Networks* in the design of a recommendation system.

When designing a recommendation system, a variety of information may be available. In most cases, we are provided with the ratings of some items by a set of users. In other cases, metadata about the users or items is also given. This information needs to be present in our graph. When we dispose of information of both users and items, we need to represent such information in a *bipartite graph*.

According to *Wikipedia*, in the mathematical field of graph theory, a bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint and independent sets. In this type of graph, the two vertices i and j of an edge (i, j) must belong to the two opposite sets. We note that the graph could even have more than two types of entities (heterogeneous graph), but bipartite graph will be considered for simplicity. However, it must be noticed that for this type of graphs, convolutions are a bit more difficult, as different types of entities are considered. More detailed information will be given in Chapter 4.

With this in mind, a general architecture as the one shown in Figure 2.15 can be used to solve this problem.

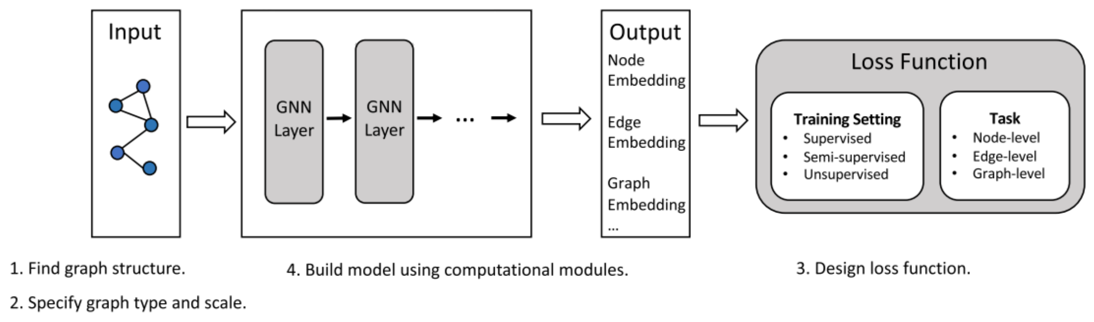

Figure 2.15: The general design pipeline for a GNN model found in [1]

According to Figure 2.15, we define first the entities (nodes) of the problem and the relationships between them. Then, several layers are applied. After that, some sort of embeddings are obtained for each node. With these embeddings, a readout MLP is defined according to the task and connected to a loss function, in order to maximize the accuracy of the embeddings.

2.5 MovieLens version specification

In order to evaluate the performance of the different approaches we will use **MovieLens 100K** [4], a dataset collected by the GroupLens Research Project at the University of Minnesota and released in 1998. This dataset consists of 100,000 ratings from 943 users on 1682 movies. The details of this dataset are the following:

- Ratings go from 1 to 5 (increasing by units of 1).
- Each user has rated at least 20 movies. However, there is no restriction on the number of ratings a movie has received, so it could be possible for a movie to have only one available rating.
- Available user meta information: age, gender and occupation.
- Available movie meta information: genres, the title and the IMDb (Internet Movie Database) link to the movie.

CHAPTER 3
Recommendation with Graph Signal Processing

In this chapter, based on the theory presented in Chapter 2, we describe the implementation of a recommendation system using Graph Signal Processing, as well as the experiments performed with the MovieLens dataset [4].

As explained in section 2.2, two approaches can be used for designing a recommendation system based on Graph Signal Processing: user-based and item-based collaborative filtering. In this chapter, we will only describe the **item-based approach** as the user-based approach basically follows the same process.

The structure of this chapter is as follows: section 3.1 overviews the objectives we pursue with a Graph Signal Processing model as well as its usage in a recommendation application; section 3.2 explains the training procedure step by step; section 3.3 describes the model architecture; section 3.4 specifies the implementation process and finally sections 3.5 and 3.6 show the experiments performed as well as the obtained results. Through the experiments, we aim to identify the best model for recommendation.

3.1 Objectives

In this project, our aim is to build a model that predicts the rating that a user would give to **one** particular movie, given the values of the movies that the user did rate. In other words, given the graph signal of a user, we will handle the recommendation of one specific movie.

The reason that the recommendation of multiple movies for a user is not addressed in this work is because it would require lots of data, memory and time. Our goal is to investigate the design of a recommendation system with Graph Signal Processing, and to this end we put the focus exclusively in one movie.

Typically, recommendation applications suggest various items (movies in our case) to the user. Since our approach is aimed at building a model for a single movie, we need to build a model for each movie in the system. Then, given a user (which is represented as a graph signal), we will run all the available movie models and predict the rating for each movie, finally recommending the movie(s) with higher prediction ratings.

We split the available users in the system into two sets: the users in the training set will be used to build the model and the users in the test set will be used to evaluate the model predictions. The training users must be seen as a set of movie viewers that represent the likes of a general population, and from whom valuable information about the movies and their relationships can be extracted.

For learning the model for a movie m, we remove from the training set those users who have not seen the movie m, thus leading to a training set exclusively composed of users who did see m. This is so because we want to learn a model that helps recommend the movie m to people who has not seen it by predicting the score of m for such people on the basis of the information extracted from people who did see the movie. The way of working with the training set is as follows: we mask the ratings of the movie m in all the users, and the objective is to approximate the scores as close as possible to the real ones.

Users of the test set are only used for evaluating the model thus ensuring model generalization. As with the training users, in the test set we consider users who have seen movie m as our purpose is to check how the predicted scores of the model differ from the real ratings.

Finally, once the model for movie m is built and evaluated, it can be used for recommending the movie to a target user. To this end, we run the model over the graph signal of the target user, who has seen movies other than m, and the model will return the predicted score of movie m for the user.

3.2 Model setup

This section details the elements needed to build an item-based collaborative filtering recommendation model based on Graph Signal Processing.

The information of the system is represented in a graph where nodes represent movies and edges between two nodes are assigned a weight which denotes the similarity between the corresponding movies. The weight of the edges is computed applying the Pearson correlation coefficient of the ratings given by the available users to the movies.

Users are represented as graph signals; that is, vectors of M components where M is the total number of movies and each component of the vector is associated with one node of the graph. Particularly, a graph signal is denoted as x_u and a signal component as $x_{u,i}$, which meaning is the rating that user u has given to movie i.

We decided to use the Pearson correlation matrix as the GSO of our graph, as it expresses similarities between the movies. However, in further experiments we will also consider the Adjacency and Laplacian matrices as well as their normalized versions.

The steps to train the model are the following:

1. Split the user graph signals into training and test. When splitting, we must ensure that both sets contain among others, graph signals of users who have rated m. This is so because on step 3, we will clear out all users who have not seen m and we want to avoid an empty set.

2. Use the training signals to compute the similarities between movies applying the Pearson correlation coefficient. All the training signals are used so as to capture the general relationships between movies.

3. We remove from the training and test sets the graph signals of users that have not rated m as we want to learn predicting the movie m to people who has not seen the movie.

4. Mask the target rating of m in all the graph signals that will be used for training and evaluation.

By training the graph signals we learn the weights of the model for movie m to make useful recommendations. For each train signal, a prediction regarding the rating of the movie m is computed through the model. The real rating is available, as we have previously masked it, so the difference between the real rating and the predicted score is used to fine-tune the model's weights through back-propagation. Particularly, we use the criterion *MSELoss* to measure the mean squared error between the predicted rating and the real rating for all the users in the training set (MSELoss).

Similarly, test graph signals are used to evaluate the model, ensuring that the model is not just memorizing results. *RMSE* (Root-mean-square deviation) is used for this purpose (RMSE).

This procedure is described graphically in Figure 3.1.

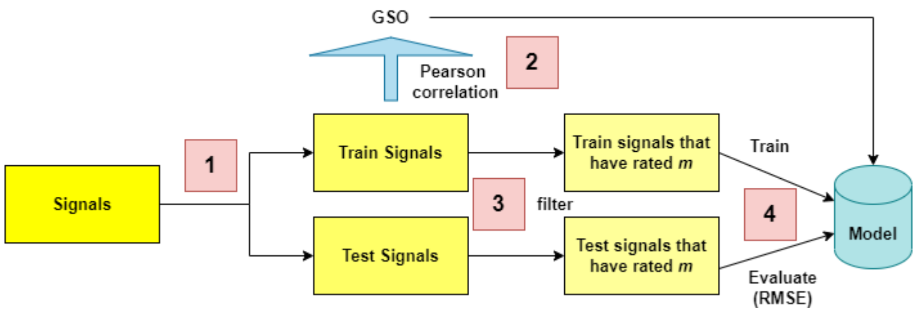

Figure 3.1: Graph signals processing for model m

3.3 Model architecture

The architecture of the process that builds the model is exposed in Figure 3.2.

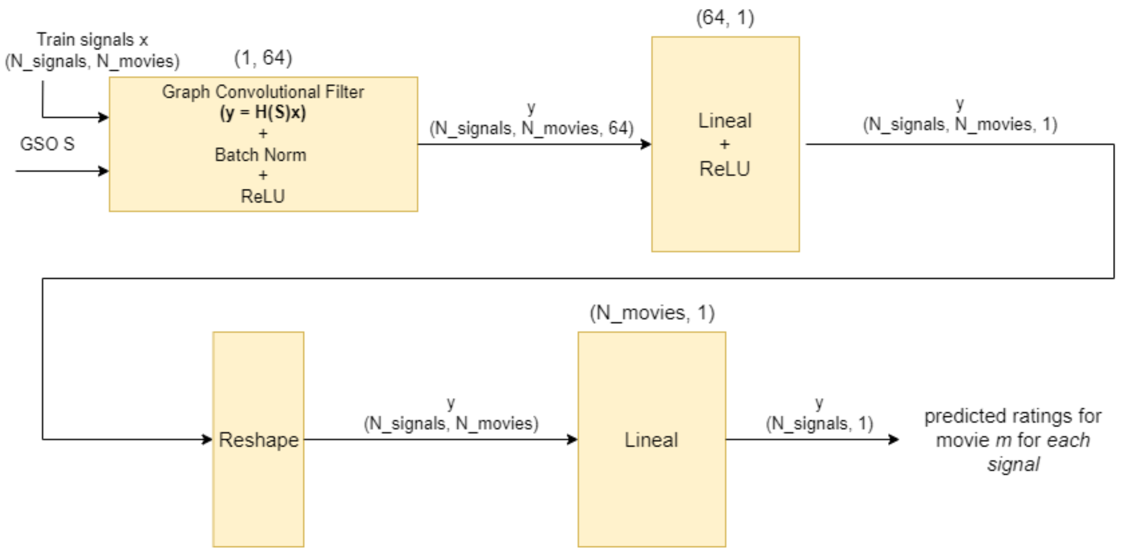
Train signals x (N_signals, N_movies)
GSO S
(1, 64)
Graph Convolutional Filter (y = H(S)x) + Batch Norm + ReLU
y (N_signals, N_movies, 64)
(64, 1)
Lineal + ReLU
y (N_signals, N_movies, 1)
Reshape
y (N_signals, N_movies)
(N_movies, 1)
Lineal
y (N_signals, 1)
predicted ratings for movie *m* for each signal

Figure 3.2: Description of model's architecture for the movie *Men in Black*

This process takes two arguments as input. The first argument is a set of graph signals of users who have seen m. This set is treated as a matrix of shape $N\_signals \times N\_movies$, where each row represents a graph signal (being $N\_signals$ the number of graph signals) and each column represents the ratings of a movie given by all the users (being

N_movies the number of movies). The second argument is the precomputed GSO S. Both arguments are used in the graph convolutional filter to create a diffused signal projection into a 64-dimension latent space. Next, we apply batch normalization to the result of the graph convolutional filter (which will normalize the range of values of the result) followed by a Rectified Linear Unit (which will clear out all negative values). Following, a series of linear layers are applied that output a scalar for each graph signal, which will be the prediction of the rating of the movie m for each signal. If the graph signals belong to the training set, *MSELoss* is computed and backpropagated to fine-tune the weights of the filter and linear layers. However, if the graph signals are from the test set, *RMSE* is computed to check the model performance.

In this process, two components can be distinguished: a *Graph Convolutional Filter* and different *Linear Layers*:

- **Graph Convolutional Filter**: This filter basically performs the operation stated in Equation 2.9. The graph signals are propagated all the needed times, saving each propagation into x. Then, we multiply x by H (the filter itself), and reshape in order to obtain the desired output. Figure 3.3 show the operations involved in the graph convolutional filter.

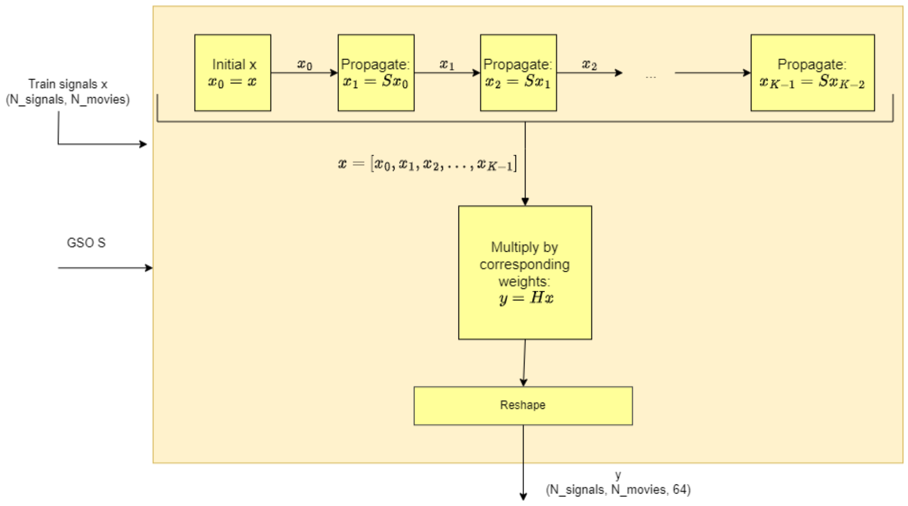

Figure 3.3: Implementation details of graph convolutional filter

  Our aim through graph filters (propagating and projecting) is to learn which neighbours have a more critical roe in order to help predict the desired rating. Actually, filters also weigh the relevance of the diffusion steps with the aim to find patterns in the latent space.

- *Linear Layers* are used to reduce dimensionality. Particularly, the first lineal layer converts the ratings from the latent space back to the original value, and the final layer simplifies the graph signal into a single number, which is the predicted rating for the signal.

3.4 Implementation details

The previously explained model has been implemented with Pytorch Geometric (*PyG*) [13], a library built upon PyTorch to easily write and train Graph Neural Networks

(GNNs) for a wide range of applications related to structured data. We will chronologically describe the implementations carried out to implement the item-based approach without going into too much detail in the construction of the system.

We remark that our aim is to train a model to predict the ratings of a movie m given the user previously stated preferences. For that, several steps have been followed.

Firstly, we convert the data that come in the form of a *csv* format into graph signals. The signals are split (in a balanced way) into training and testing sets.

Once the data is partitioned into two sets, we compute the movie similarities via the *Pearson* correlation matrix using the training set. Pearson correlation computes a value between -1 to 1: a value of -1 means a total negative linear correlation, 0 no correlation, and $+1$ means a total positive correlation. In our case, the edges of the graph are labeled with the positive Pearson correlations between movies of the training set users. This is because positive correlations indicate similar trends between variables (*If I like this movie, someone else will likely also like this movie*). Negative correlations, on the other hand, have the opposite meaning. We are not concerned with negative values as they represent conflicting tastes which is out of the scope of this work.

For each node, only the most similar neighbors are considered (5 nodes by default). In case of using the *Adjacency* matrix as GSO, it suffices to construct it from the found graph similarities, setting to 1 the matrix elements corresponding to pairs of movies with a positive correlation. Similarly, we can use the *Laplacian* matrix or any normalized matrix version. All of them were implemented and the results are shown in section 3.5 .

To implement the model training, a convolution was created with the module of *PyG* to create particular convolutions. This convolution represents the *Graph Filter Convolution* of Figure 3.3. All details of implementation are available at GitHub.

3.5 Experiments definition

We carried out several experiments in order to evaluate the performance of this approach. Specifically, we used the measure RMSE as it is popularly used in the evaluation of Recommendation Systems.

We used three movies to test the behaviour of the models: *Men in Black (1997)*, *Toy Story (1995)* and *Scream (1996)*. To compute graph similarities, a total of 802 signals were used:

- For the movie *Men in Black*, 263 signals are used for training, and 40 for testing.
- For the movie *Toy Story*, 380 signals are used for training and 72 for testing.
- For the movie *Scream*, 413 signals are used for the training and 65 for testing.

Note that graph signals from training and testing correspond to users who saw and rated the target movie. A maximum of 100 epochs were used in all the experiments as models usually converge very fast.

The first experiment consists in training and testing a model for each movie with the default parameters in order to get a baseline of the RMSE. By default, a neighborhood of 5 nodes is used, 5 filter taps are applied and data is projected into a 64-dimension latent space. We used the Adam optimizer with learning rate of 0.005. Also, technique *ReduceLROnPlateau* with patience 20 is being used for training. This technique dynamically reduces the learning rate when the *RMSE* has stopped improving for 20 epochs. This technique helps approach a minimum error.

In order to check and analyze the potential of this approach, we also tested the model as a classifier. To do so, we changed the last layer of the model shown in 3.2 to predict not only a number, but a vector of 5 elements since there are 5 classes to predict (1-star, 2-star, 3-star, 4-star and 5-star movies). Each element of the vector will represent the probability that the predicted score for the movie is equal to the corresponding class given the graph signal. For the probabilities to be normalized, we add a *Softmax* layer.

Additionally, some experiments were carried out in order to perform some sort of fine tuning regarding the Graph Shift Operator. To this end, we considered different levels of sparsification (number of neighbours of each node), different number of filter taps and different number of dimensions of the latent space.

Finally, a weighted *MSELoss* will be tried out in order to check if better results can be accomplished.

In summary, the series of experiments carried out in this project are:

1. Basic approach with default parameters.

2. Problem as classification.

3. Graph Shift Operator and sparsification level

4. Number of filter taps K and dimension projection

5. Weighted *MSELoss*

3.6 Results

This section shows the results for the experiments described in the previous section. The results correspond to the median of three runs.

3.6.1. Basic approach with default parameters

Results for this approach are shown in the following table:

Movie	Train RMSE	Test RMSE
Men in Black	0.89	0.945
Toy Story	0.88	1.01
Scream	0.80	1.03

Table 3.1: RMSE results for the item-based approach with default parameters

A RMSE of 0.95 for *Men in Black*, 1.01 for *Toy Story* and 1.03 for *Scream* is accomplished, which can be interpreted as accurate results. The evolution of the RMSE through the training of the three models is shown in the following figures.

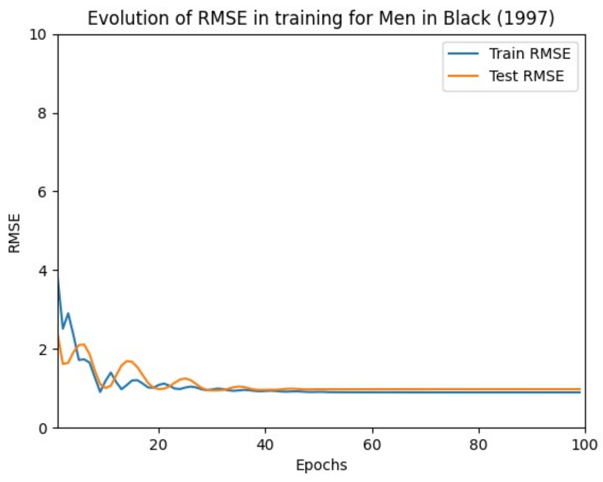

Figure 3.4: Evolution of RMSE in *Men in Black* training

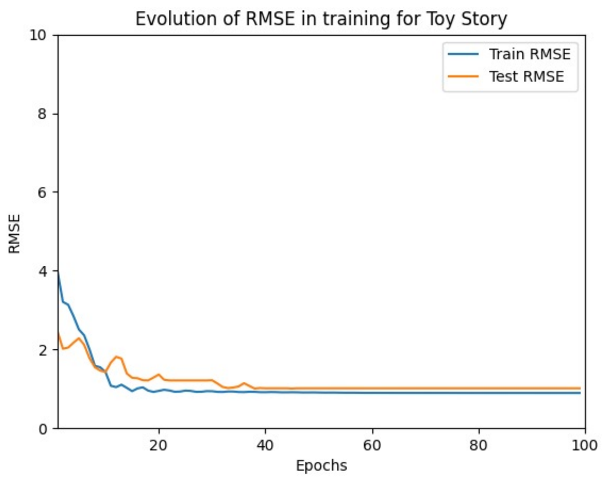

Figure 3.5: Evolution of RMSE in *Toy Story* training

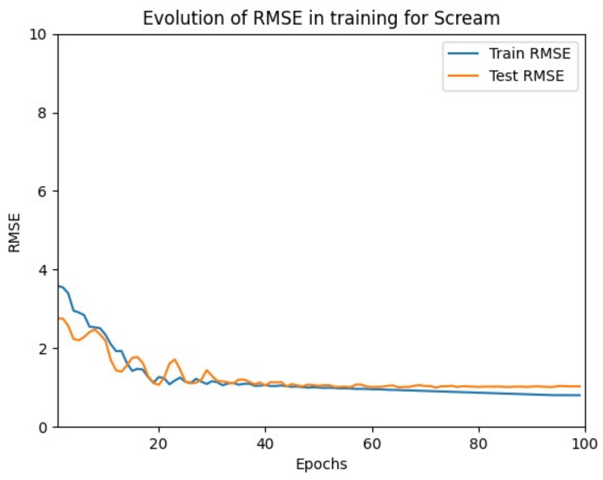

Figure 3.6: Evolution of RMSE in *Scream* training

The model for *Men in Black* (Figure 3.4) and *Toy Story* (3.5) converge on 40 epochs. The model for *Scream* (Figure 3.6) does not fully converge, but the value of RMSE starts decreasing very slowly from epoch 60.

Looking at the predicted ratings, the model for *Men in Black* learns a score between 3 and 4 (mainly 3.7) in order to minimize the RMSE. The effect of using *MSELoss* is that to minimize the loss, it always predicts a value around 3.7 as this is the mean of the distribution of the ratings of the movies. This is because *MSELoss* penalizes large errors, so why should the system risk to predict 5 or 1 and make a very big mistake when 3 is the mean? The same behaviour is observed in the *Toy Story* model and in the *Scream* movie.

The system is not able to learn the proper triggers for the rating prediction. Even though we get a low RMSE value, this model is not useful for recommendation. Some additional trials were made to overcome these limitations:

- **Lower the learning rate**: We experimented with a learning rate lower than 0.005 but it turns out we got the same results at a slower convergence.

- **Early stopping**: during the training process, the system learns on every epoch which predictions are more helpful to minimize the RMSE. Several epochs lead the model to predict a value between 3 and 4 as it interprets this is the optimal value to minimize the loss.

  In this experiment we stopped the training before the system realizes of an optimal value between 3 and 4 for reducing the RMSE. Our intention is to find out if an early stop allows us to get a variety of ratings rather than consistently a value around 3.7.

  The results showed that the predictions obtained with the initial epochs accomplish a more diverse recommendation. However, it turns out that the RMSE is higher both in training and testing. Particularly, for *Men in Black*, around epoch 10, the RMSE is 1.16 in training and 1.5 in testing, and the predicted ratings are not that homogeneous. On epoch 12, *Toy Story* accomplishes a RMSE of 1.61 in training and 1.55 in testing, but it returns ratings that vary from 1 to 5. The same happens in *Scream* on epoch 20, with a value of RMSE in training of 1.55 and 1.41 in testing. The statistics regarding the percentage of appearance of predictions are: rating 1 is predicted 2.76%, rating 2 is predicted 14.36%, rating 3 is predicted 33.14%, rating 4 is predicted 39.77% and rating 5 is predicted 9.97%.

  Summarizing, stopping the system before convergence results in a proper approach to obtain a variety of ratings. The model is trained fewer epochs, it does not overfit the training but it returns a higher RMSE. As a result we need to find at which epoch there is a balance between RMSE and diversity of ratings in order to obtain an acceptable recommendation system.

- **Transform the problem into a classification problem**: using RMSE in regression problems is hard because the model can lead to tricky local minima results. We can transform the regression problem into a classification problem so that the system classifies signals into five classes (1, 2, 3, 4 and 5), each corresponding to a number of stars that movies can receive. This will be studied in next subsection.

We also tested a user-based approach obtaining the same behaviour but with higher values of RMSE. This is due among other reasons to the fewer available training signals.

Next experiments will only be carried out for movie *Men in Black*, as the same behaviour has been proved regardless of the target movie.

3.6.2. Classification problem

In this experiment we transform the prediction problem into a classification problem in order to check if more information could be learnt. It must be noticed that *CrossEntropy-Loss* will be used instead of *MSELoss*.

The results of the classification will be measured through accuracy instead of RMSE. That is so because RMSE is a performance measure, and it is mostly used for regression problems. For classification, accuracy is a more appropriate measure. In fact, the aim of transforming the problem from regression into classification is to train using CrossEntropyLoss to check if better results could be achieved. CrossEntropyLoss is commonly used to quantify the difference between two probability distributions, whereas RMSE is the standard deviation of the prediction errors.

Results are shown in the Table 3.2, alongside with the evolution of the accuracy during the training (Figure 3.7):

Approach	Train Acc	Test Acc
Men in Black model with classification	0.93	0.275

Table 3.2: Accuracy results for the *Men in Black* model with classification

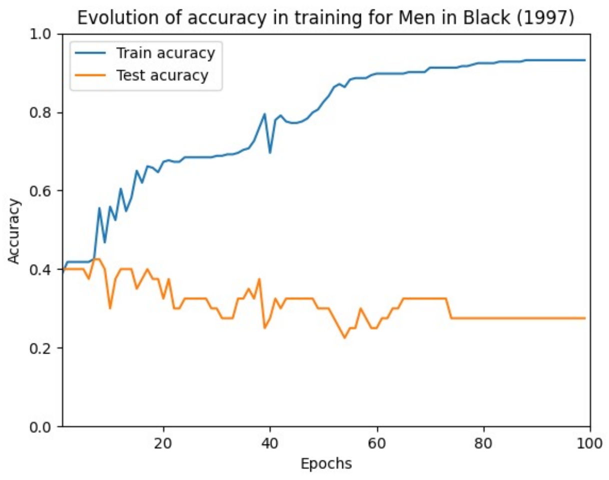

Figure 3.7: Evolution of accuracy for *Men in Black* model with classification

As we can see, even though the accuracy in training is higher than 90%, in testing it hardly passes the 25%. Results suggest that the model is not learning, but memorizing the training data, phenomenon known as overfitting. The training phase is not capable of finding any pattern in the data which might be due to the sparsity of the graph signals. In general, the model is not appropriate for our purposes, as it simply memorizes.

No experiments have been made with the user-based approach, as this approach has even fewer signals and the same behaviour is expected that the one observed on item-based classification.

All in all, even though the regression approach had its issues, it accomplished better results. For that reason, from now on experiments will be done with the original regression approach but limiting training epochs to 10.

3.6.3. Graph Shift Operator and sparsification level

In this section, our aim is to check the Graph Shift Operator that works best on the model for the movies *Men in Black*, *Toy Story* and *Scream*, as well as how the number of considered neighbors affect the value of RMSE for each movie.

We will evaluate the results obtained in training and testing for 10 epochs with learning rate of 0.005 and considering only 5 best neighbors. The same partitions will be used in order to assure available data is equal for every run. Results are shown in next table:

Movie	GSO	Train RMSE	Test RMSE
Men in Black	Pearson	0.90	**1.05**
	Adjacency	2.19	1.90
	Adjacency Normalized	1.38	1.58
	Laplacian	**0.88**	2.23
	Laplacian Normalized	1.34	1.12
Toy Story	Pearson	**1.25**	1.46
	Adjacency	1.85	1.98
	Adjacency Normalized	1.52	**1.19**
	Laplacian	1.71	1.59
	Laplacian Normalized	1.56	1.57
Scream	Pearson	**1.48**	1.62
	Adjacency	2.1	1.78
	Adjacency Normalized	2.13	**1.60**
	Laplacian	1.84	1.87
	Laplacian Normalized	1.66	1.67

Table 3.3: RMSE results depending on the GSO for the used and the target movie

As we can see, using Pearson as GSO in testing (the one we have been using so far) is the measure that works best on *Men in Black*. However, Adjacency Normalized is the one that achieves better performance on testing for both *Toy Story* and *Scream*. The Laplacian also achieves best RMSE in training for *Men in Black*, but we can observe that it helps the system over-fit faster in testing. Using the Laplacian normalized, also allows for low values of RMSE.

As a conclusion, both Pearson and the normalized versions of Adjacency and Laplacian seem to be an acceptable GSO for this problem.

Following, we will vary the number of neighbors to be considered in order to check if there really exists a trade-off: more neighbors consume more time and resources but accomplish better results or it just add noise to the prediction? Results are shown in the following table:

Movie	Neighbors	Train RMSE	Test RMSE
Men in Black	5	**0.90**	**1.05**
	10	1.51	1.80
	50	1.21	1.25
	200	2.77	2.10
	500	2.68	2.60
Toy Story	5	**1.25**	1.46
	10	1.34	**1.39**
	50	1.72	1.55
	200	2.71	2.38
	500	2.98	2.81
Scream	5	**1.48**	1.62
	10	1.78	1.98
	50	1.88	**1.18**
	200	1.96	1.63
	500	2.67	2.75

Table 3.4: RMSE results depending on the neighbors considered and the target movie

The three target movies accomplish the lowest train RMSE using 5 neighbors. It makes sense that if the system manages less information per node, it learns the train faster. Nevertheless, we want the system to predict accurate results for unseen signals, so we will focus on test RMSE. The best test RMSE is accomplished with 5 neighbors for *Men in Black*, 10 neighbors for *Toy Story* and 50 neighbors for *Scream*.

It looks like between 5 and 50 neighbors is good representation of the data depending on the particular signals of the movie. However, if more neighbors are taken into account, some sort of noise is generated, and this will hinder the system to find patterns between all the connections.

3.6.4. Number of filter taps K and dimension projection

Finally, a study on the best parameters will be carried out in order to check if some sort of improvement can be achieved. Again, movies *Men in Black*, *Toy Story* and *Scream* with default parameters will be tested.

Firstly, we study the impact of the number of filter taps (see Table 3.5). According to the table, an ideal number of filter taps is around 5. A very low number of diffusions will not help the system to learn the embeddings, but too many is also not helpful for the model, as results are worse.

Movie	K (Filter taps)	Train RMSE	Test RMSE
Men in Black	2	1.61	1.21
	5	**0.90**	**1.05**
	10	2.28	2.00
	25	2.6	2.57
	50	0.99	2.75
Toy Story	2	2.92	1.56
	5	1.26	**1.46**
	10	1.29	1.97
	25	1.73	1.55
	50	**1.01**	1.96
Scream	2	2.07	1.62
	5	**1.48**	**1.15**
	10	2.11	1.58
	25	2.46	2.04
	50	1.56	1.78

Table 3.5: RMSE results depending on the filter taps and the target movie

Regarding the latent space dimensions where the data is projected, the results can be observed in Table 3.6. A dimension around 32 would be a good number. Again, a lower dimension may not be sufficient, and a higher dimension may be too complex for the system to find a pattern.

Movie	Latent space dimensions	Train RMSE	Test RMSE
Men in Black	256	2.62	2.87
	128	2.17	2.43
	64	1.92	1.92
	32	**1.46**	**1.61**
	16	2.05	1.67
Toy Story	256	2.37	1.87
	128	1.60	1.26
	64	1.26	1.46
	32	1.33	**1.10**
	16	**1.19**	1.78
Scream	256	2.29	2.59
	128	1.24	1.38
	64	1.48	1.62
	32	**1.18**	**1.21**
	16	2.11	1.60

Table 3.6: RMSE results depending of the dimensions of the latent space and the target movie

3.6.5. Weighted *MSELoss*

In this last experiment, we used a weighted version of *MSELoss* in order to avoid homogeneous predictions. Weighted *MSELoss* gives more importance to errors from predictions that are less common like 1 or 5. With the unweighted *MSELoss*, the system tends to learn predictions around 3.7 because this value is close to the mean of the distribution of the ratings. With the new loss function, predicting a wrong 1 or 5 has a large penalty, making the error even bigger and thus the system gives more importance to predict the right numbers instead of the mean.

For this final experiment, we will show the results from the best model out of three trials. As some experiments have been carried out previously, information obtained from those is going to be applied. In this experiment, we will project into a 32 dimensional latent space, as it has proven to be more effective. Pearson, normalized Adjacency and normalized Laplacian will be used as GSO, as they have shown to be the most effective GSO for this problem. The rest of the parameters will be used with the default value. The results for each option of GSO are shown in Table 3.7, Table 3.8 and Table 3.9.

- Pearson:

Movie	Train RMSE	Test RMSE	Rating distribution
Men in Black	1.54	1.34	2(2.5%),3(70%),4(17.5%),5(10%)
Toy Story	1.46	1.25	2(2.77%),3(70.83%),4(20.83%),5(5.57%)
Scream	0.73	1.06	1(4.5%),2(8.1%),3(30.7%),4(49%),5(7.7%)

Table 3.7: RMSE results using weighted RMSE with Pearson GSO

- Normalized Laplacian:

Movie	Train RMSE	Test RMSE	Rating distribution
Men in Black	0.81	1.26	3(62.5%),4(25%),5(12.5%)
Toy Story	0.43	1.24	2(6.95%),3(26.38%),4(56.95%),5(9.72%)
Scream	0.69	1.0	2(10.76%),3(49.25%),4(26.15%),5(13.84%)

Table 3.8: RMSE results using weighted RMSE with Laplacian Normalized GSO

- Normalized Adjacency:

Movie	Train RMSE	Test RMSE	Rating distribution
Men in Black	1.50	1.36	3(72.50%),4(15%),5(12.5%)
Toy Story	1.46	1.36	2(19.44%),3(52.77%),4(16.68%),5(11.11%)
Scream	0.71	0.95	2(7.71%),3(52.30%),4(32.30%),5(7.69%)

Table 3.9: RMSE results using weighted RMSE with Normalized Adjacency GSO

As we can see in the tables, low RMSE values are obtained without giving up on diversity on the obtained predictions. For the movies *Men in Black* and *Toy Story*, normalized Laplacian is the GSO that alongside the weighted *MSELoss* accomplishes the lowest RMSE in both training and testing. For the movie *Scream*, normalized Laplacian is the one obtaining lowest train RMSE but normalized Adjacency is the one accomplishing better results on testing.

It must also be noticed that Pearson GSO is the only alternative that predicts a value of 1. We hypothesize that a 1-star score is a very uncommon rating, and thus it is difficult for the system to learn how to handle it. It could be possible that with a bigge r set of signals, the system was capable of obtaining even lower RMSE.

CHAPTER 4

Recommendation with Bipartite Graph

In this chapter, we describe the implementation of a recommendation system using bipartite-graphs through spatial-based GCN based on the theory exposed in section 2.3 as well as the experiments performed with the Movielens dataset [4].

The structure of this chapter is as follows: section 4.1 overviews the objectives we pursue with a Spatial-based GCN recommendation systems as well as its usage in a real applications; section 4.2 explains the training procedure step by step; section 4.3 describes the model architecture; section 4.4 specifies the implementation process and finally sections 4.5 and 4.6 show the experiments performed as well as the obtained results. Through the experiments, we aim to identify the best model for recommendation.

4.1 Objectives

In this approach, we will use a bipartite graph, an undirected weighted graph that is composed of two different types of nodes (users and movies). In this graph, weighted edges connect nodes of type *user* with nodes of type *movie*. An edge between a user and a movie means that the user has seen and rated the movie. The rating that the user has given to the movie is encoded as the weight of the edge connecting both entities. No connections of type user-user or movie-movie will be considered in this implementation.

The task to solve in this approach consists in, assuming an edge exists between two nodes, predicting the weight of such edge. Unlike the common link prediction in GNNs, our task is not predicting whether or not a link exists between two nodes but to assume the edge existence and quantify its weight. Therefore, our aim is to build a model which predicts the weights of target links. The model will learn relations from the ratings given by the users to the movies using both the initial features of the nodes and a set of edges which weight we already know. Briefly, several steps can be identified:

1. **Train and evaluate a model**: The initial features of the nodes as well as a bipartite graph with the available ratings will be used to learn the relationships between the different users and movies in the system.
2. **Recommendation**: In order to recommend one movie from the available set of movies present in the system to a target user a, we use the previously trained model. To this end, the input to the model will be the initial feature vectors of nodes and another bipartite graph. This graph will consist of the same nodes than the graph used for learning the model but with a different set of edges since in recommendation an

edge denotes the rating that the target user **would** give to a movie. Therefore, an edge will be present between the user and all the non-seen movies by the user (as these are the edges whose weight we wish to predict). The model will predict for each edge an specific weight from 1 to 5, and the movie with the highest weight will be recommended to the user.

4.2 Model setup

This section describes the elements needed to build a recommendation system based on a bipartite graph.

Let us briefly remember the available information described in section 2.5. We have information about users, movies and their interactions. The knowledge available about users is the age, the gender and the occupation, whereas for the movies we have the genres, the title and the IMDB link. Interactions are of the type user-movie and they consist of the rating that the users have given to the movies they have seen.

The first thing to do is to represent the information in a bipartite graph which is formed by two types of nodes: users and movies. Both types of nodes will be associated to different information (feature vectors) but every node of the same type will have the same information.

- **Nodes of type *movie***: the genre and the title make up the encoding of the initial feature vectors of nodes of type movie. Genres will be managed as one-hot encoded vectors, and the title will be transformed into a vector using a pre-trained transformer. The final feature vector of a movie will be the concatenation of its one-hot encoded genres with the sentence embedding associated to its title.
- **Nodes of type *user***: By default we will not use information about the users other than a number to identify them. The initial feature vector of a user is a one-hot encoding of the number of users.

Once the nodes and their initial feature vectors have been specified, we must define the links between nodes. A link will exist between a user and a movie if the user has seen and rated the movie. The rating will be reflected on the edge weight. In this approach, we aim to predict the edges weights. Figure 4.1 shows all the previously mentioned aspects.

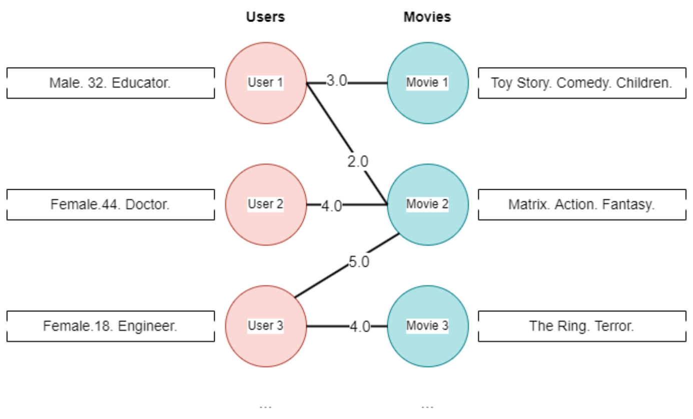

Figure 4.1: Bipartite graph showing available information

Users and movies are connected through links, and each nodes has its associated initial feature vector. Notice that Figure 4.1 shows initial feature vectors in natural language in order to ease the understanding, but in reality, it would be a vector encoding that represents that information.

A set of the edges from this graph will be used for training the model, another set for validation and another set for testing. The steps to train the model are:

1. Definition of the initial feature vectors of the nodes.

2. We split the edges into training, validation and testing sets, and mask its respective weights.

3. Training of the model. This process can be further subdivided in different steps:

   (a) The model receives the initial feature vectors of the nodes as well as the training edges with masked weights.

   (b) The system propagates the feature vectors of the different nodes a specific number of times in order to generate node embeddings.

   (c) Compute an embedding for each edge. An edge embedding is the concatenation of its respective user embedding and movie embedding.

   (d) Transform the edge embedding into a number through a MLP. This number will represent the predicted weight for the edge.

   (e) Compute *MSELoss* of the learnt weights with respect to the real ones and backpropagate.

   This process will be repeated a specific number of epochs. At the end of each epoch, *RMSE* will be computed with validation edges. If this metric does not improve in 20 epochs, the learning rate will be reduced in order to search for a local minima.

4. Evaluate the system. Similarly, test links will be used to evaluate the model using the *RMSE*.

4.3 Model architecture

Figure 4.2 shows the architecture of the bipartite graph approach to train the model.

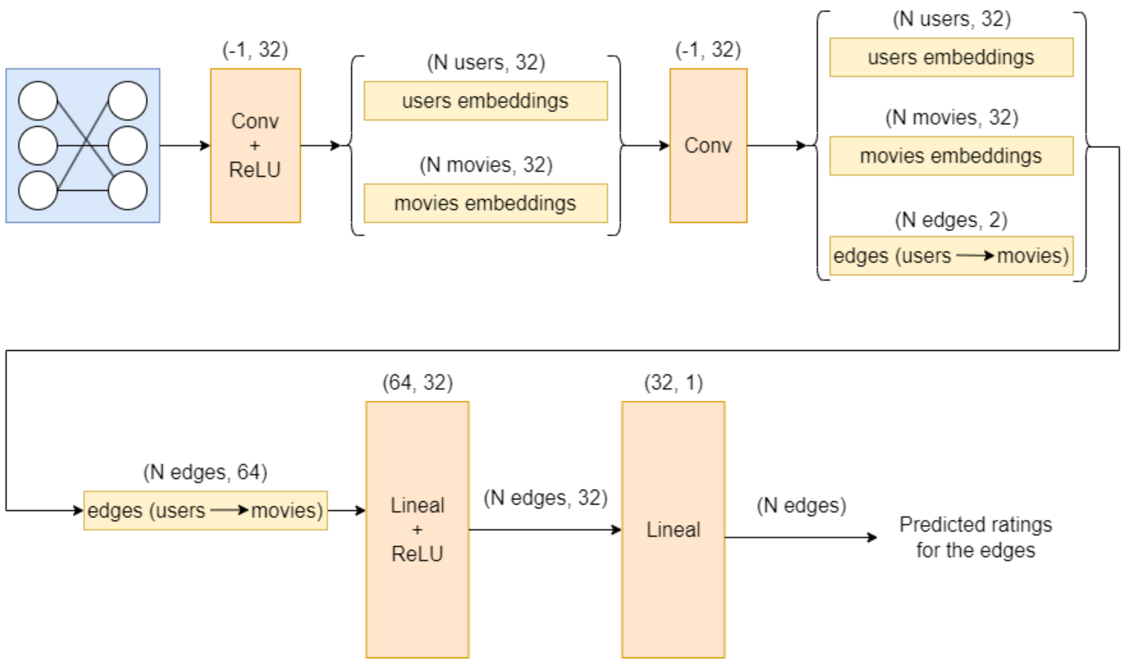

Figure 4.2: Bipartite graph architecture

As we can see, the input of the model is the graph, with its feature vectors and training links. The first convolution transforms the initial feature vectors into an embedding of a 32-dimension latent space. The same type of convolution is applied twice. After this, we end up with 32-dimension embeddings of users and movies.

Following, the edges are encoded as a concatenation of the embeddings of the user and movie they connect, becoming 64-dimension edges. Finally, two linear layers are concatenated to pass from 64 dimensions to 1, which will be the predicted rating for the edge.

Different types of convolutions can be defined. The most important ones in this project are *SAGEConv* and *GAT*.

- **SAGEConv** performs the *GraphSAGE* operator introduced in [14], where embeddings are computed following equation 4.1:

$$h_i^k = W_1 h_i^{k-1} + W_2 mean_{j \in \mathcal{N}(i)} h_j^{k-1} \tag{4.1}$$

  As we can see, node i is updated with the mean of the embeddings of its neighbors and itself. This equation is equal to Equation 2.10, as it uses the same aggregation and update functions.

- **GATConv** stands for the graph attentional operator [15]. Embeddings in this case are computed adding attention coefficients α, as shown in Equation 4.2:

$$h_i^k = \alpha_{i,i} \Theta h_i^{k-1} + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \Theta h_j^{k-1} \tag{4.2}$$

  where attention coefficients α come defined in 4.3, and $||$ represents a concatenation:

$$\alpha_{i,j} = \frac{exp(LeakyReLU(a^T[\Theta h_i^{k-1} || \Theta h_j^{k-1}]))}{\sum_{z \in \mathcal{N}(i) U \{i\}} exp(LeakyReLU(a^T[\Theta h_i^{k-1} || \Theta h_z^{k-1}]))} \tag{4.3}$$

The idea is to compute the hidden representations of a node in the graph by processing its neighbors through a self-attention strategy.

An overview of some widely adopted convolutions in the field of recommendation such as *SageConv*, *GATConv*, *GatedGraphConv*, *HypergraphConv*, etc. is presented in [16]. In our work, we will only only use *SageConv* and *GATConv* as the others are not yet adapted in *PyG* to be handled in bipartite graphs.

Remember that whatever convolution we use in the system is taking into account both types of nodes (users and movies), so in order to accomplish the same behaviour, messages are duplicated and aggregated for the different types of nodes, once they have been projected into the latent space.

4.4 Implementation details

In this section, our aim is to specify some details of the implementation of the recommendation approach with bipartite graph using *PyTorch Geometric*.

First, a graph like the one specified in Figure 4.1 is created. For the nodes of type *movie*, the sentence of the title is transformed in order to obtain a meaningful representation. Particularly, we used *all-distil-roberta-v1* [17], a transformer which maps sentences and paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search. Then, we concatenate the title encoding and a one-hot encoded vector that represents the genres of the movie. The concatenation of both information (representation of the title through a sentence embedding and the one-hot encoded genres) makes up the initial feature vectors of the nodes of type *movie*. As for nodes of type *user*, by default we assume that there is no available information other than a number that identifies the user. Therefore, the initial feature vector of *user* nodes is a one-hot encoding of the user identifiers.

The handling of the bipartite graph (as it has two different types of entities) has been facilitated with *HeteroData*, an object provided by *PyTorch Geometric* that facilitates the definition of nodes and edges of different types, for a further automatic process. In this object, we save all the embeddings, edges and weighs, resulting in a *.pt* file that will contain all the necessary information of our data and graph.

Once the graph his created, the data is split in training, validation and testing sets. We used the function *RandomLinkSplit* to randomly separate the links into training (80%), testing (10%), and validation (10%). Therefore, $80,000$ links are used to train, 1000 to validate and 1000 to test (as we are provided with $100,000$ ratings).

Following, we define the two parts of the model, an encoder and a decoder. The encoder is in charge of applying two convolutions (*SAGEConv*) whereas the decoder is in charge of the final readout layer or MLP. Each convolution infer the input size and map it into a vector space of 32 dimensions. We must notice that the handling of the bipartite graph is done in an automatic way through *PyTorch Geometric*, as we only need to pass the encoder to a function called *to_hetero* in order for the model to understand that we are working with a heterogeneous graph and apply the necessary operations. The model trains with the *MSELoss* computed with predicted and expected link weights in the training set.

The code is available at GitHub.

4.5 Experiments definition

We carried out some experiments in order to evaluate the performance of the bipartite graph approach. As standard in recommendation systems, RMSE will be used to evaluate the model. Partitions defined for training, validation and testing will be common throughout all the experiments, having $80,000$, $10,000$ and $10,000$ links each respectively. A maximum of 200 epochs will be used.

The first experiment is about changing the initial embeddings for both type of nodes in order to check the information that is more useful or what information turns out to be rather noisy than being helpful in predicting the link weights. We will also experiment with the information related with the user, considering age, gender and occupation.

Once this is done, we carried out further experiments to check which type of convolutions accomplish better performance, as well as some sort of analysis of the observed behaviour. Finally, the projection space will also be tested.

In summary, the list of experiments is:

1. Best nodes embeddings
2. Convolutions
3. Dimensions projection

4.6 Results

In this section we show the results obtained for the bipartite experiments described in section 4.5 will be shown. The results correspond to the median of three runs.

The baseline model applies two *SAGEConv* convolutions that project node embeddings into a 32-dimensional latent space followed by two linear layers. We used the Adam optimizer with learning rate of 0.01 for training, and the technique *ReduceLROnPlateau* with patience 10 in validation.

4.6.1. Best nodes embeddings

This experiment is about checking the relevant information in the nodes representation. To this end, we present the list of the experiments we have done for this purpose in the following Table:

ID	Movie's embedding	User's embeddings	Train RMSE	Val RMSE	Test RMSE
1	one hot categories + title sentence embedding	no info	0.96	0.99	0.98
2	one hot categories + title sentence embedding	user age, gender and occupation (one hot encoded)	1.08	1.08	1.08
3	title and categories sentence embedding	user age, gender and occupation in sentence embedding	1.07	1.09	1.08
4	no info	user age, gender and occupation (one hot encoded)	0.99	1.03	1.01

Table 4.1: RMSE results depending on the representation used for the nodes

The evolution of the RMSE in the training phase is shown in Figure 4.3. As we can see, the model converges fast but keeps improving until epoch 175. Another aspect to be mentioned is that the predicted ratings do not converge to 3.5 this time. There exist diversity in the predictions.

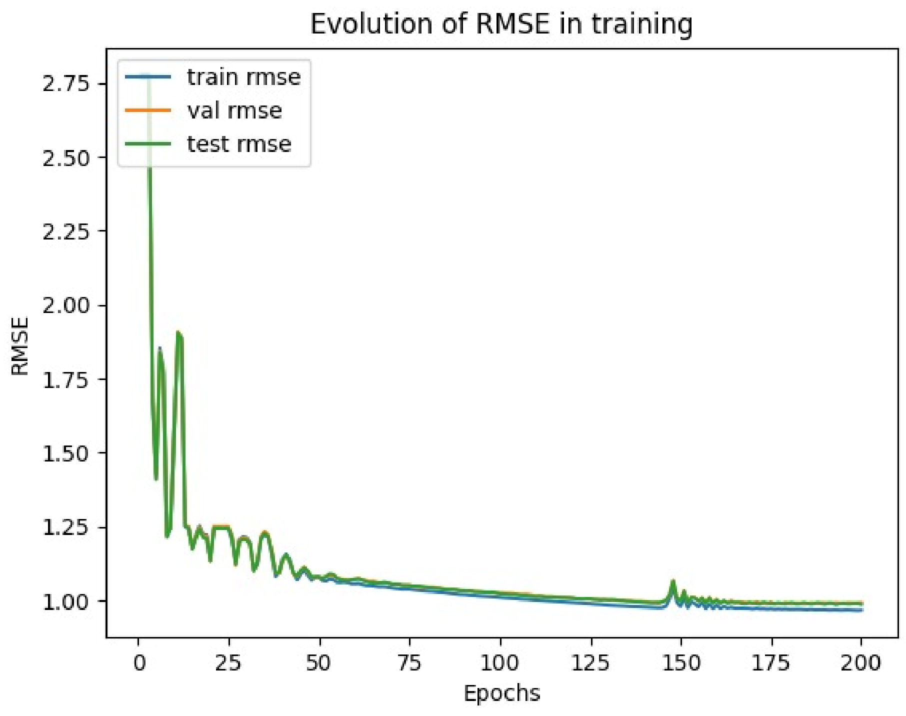

Figure 4.3: RMSE evolution in training phase for experiment ID 1

The results show that the model performs best when we use the default embeddings (ID 1 of Table 4.1): using one-hot categories for the movies concatenated with the sentence embedding of the title and no information of the user. This is an unexpected result, as we somehow believed that the variable age would be an influential factor to make a recommendation. However, experiments with IDs 2 and 3 of Table 4.1 show that having information of both movies and users can be a bit redundant for the training process.

Maybe the system is not able to learn useful embeddings for the users and movies having that much information.

Regardless diversity, unlike early GSP results, predicted numbers have been shown to be quite varied. We believe that this may be due to the usage of meta information about users and movies along with their interactions. As more information is considered, the system focuses on learning how to create node embeddings and how to decode them into ratings. However, is important to find a balance in the amount of information used, as it looks like too much information could end up hindering the learning process.

4.6.2. Convolutions

Using the baseline model, we tried out some different convolutions. Unfortunately, some convolutions that seem popular in recommendation systems mentioned in [16] are not adapted to work with bipartite graphs. This way, only those that are available for bipartite graphs were tested.

From the available convolutions adapted for bipartite graphs, most of them did not throw promising results. Only the three convolutions presented in Table 4.2 output good results.

Convolution	Train RMSE	Val RMSE	Test RMSE
SAGEConv	0.96	0.99	0.98
GATConv (heads=1)	1.02	1.04	1.02
GATConv (heads=8)	1.00	1.02	1.01

Table 4.2: RMSE results depending on the convolution used

As we can see, the convolution that accomplishes better performance is *SAGEConv*, followed closely by the *GATConv* with 8 heads. *SAGEConv* has proven not only to be the most effective, but the most straightforward, as time consumed per convolution is less than 1 second. This fact proves that simpler computations can lead to better overall performance, rather than sophisticated and complex convolutions (at least for this specific dataset and task).

4.6.3. Dimensions projection

Finally, we also tested different latent space projections so as to check if an improvement of the RMSE could be achieved by using a different dimension for the latent space.

Dimensions projection	Train RMSE	Val RMSE	Test RMSE
16	0.95	0.98	0.98
32	0.96	0.99	0.98
64	0.97	1.00	0.98
128	0.97	1.00	0.98
256	1.22	1.22	1.20

Table 4.3: RMSE results depending on the hidden channels dimension

It appears that the projected dimension accomplishes a lower RMSE in training than validation or testing. Lower dimensions help ease the training, building easier embeddings to decode whereas higher dimensions like 256 make it difficult for the model to learn since too many dimensions can lead to complex node representations.

CHAPTER 5

Conclusions and future work

In this section, our objective is to highlight some of the lessons learned during this project, comparing and evaluating the usage of the different approaches.

5.1 Comparison and discussion of results

In this project, two different approaches have been used and both share some common aspects. For example, both have common characteristics related to GNNs like the usage of a graph or the idea of propagating information through neighbors. However, there are some notable differences between them:

- GSP tries to leverage similarities on the ratings of the movies to predict the behaviour of a generic user. However, the bipartite approach propagates both the user and movie features, so it can find patterns in the ratings that the users have given to the movies.
- GSP learns the underlying structure of the graph by doing an Eigen decomposition of the GSO. However, the bipartite graph approach understands the features of a node based on its local neighbours by propagating information a particular number of times.
- Target users are quite different. GSP recommendation focuses on a generic user whereas the bipartite approach learns the interactions between specific users and movies.
- The specific task to solve is also completely distinct. In GSP, the task could be seen as node classification, as nodes of the graph represent movies and our aim is to predict a number from 1 to 5 for the node related to the target movie. However, in bipartite approach, the task could be simplified as predicting weights of hypothetical links.

Following we comment on the obtained results. Regarding the spectral-based GCN based on GSP, three movies have been exhaustively tested in order to generalize the results of the experiments. Experiments accomplished good RMSE in all of them but the model tended to converge to one same rating. In order to fix this issue, early stopping was found to be one solution to avoid the convergence in exchange for a higher RMSE. Another solution to solve this issue was to use a weighted *MSELoss* that gives more importance to errors from unusual ratings. This is the approach that worked best on our movies, accomplishing a 1.26, 1.24 and 0.95 in test RMSE for *Men in Black*, *Toy Story* and

Scream, respectively. It must be noticed that this RMSE is only accomplished for the prediction of one movie.

On the other hand, for the spatial-based GCN based on a bipartite graph, the approach was able to obtain a RMSE lower than 1 (0.98) for the whole system. This approach converges and provides recommendations of all kinds.

All that said, we consider that both approaches could be useful depending on the information and the available data. No conclusions can be extracted about which one accomplishes a better recommendation, as GSP focuses on a target movie each time and the other approach has a more general view of the system. However, we hypothesize that both approaches could improve the RMSE by training with more data, as at the end of the day, they are neural networks. Finally, and speaking about their usage, we believe that while bipartite could be really useful in systems with lots of information about entities (users and items), GSP approach could be an alternative to those systems which we only dispose of user interactions with products, and where we do not want to recommend to a specific user, but to a generic one.

5.2 Conclusions

At the beginning of this report, a series of objectives have been defined. In this section, our aim is to evaluate their state.

The first objective was to understand the underlying theoretical concepts of Graph Neural Networks. We consider that this objective has been covered, as theory has been explained in a simple and structured way, reviewing all the learnt aspects.

The second and third objectives were the building of two recommendation systems using two specific types of GNNs, and evaluating the models. For that, we have built two recommendation systems following two different approaches (GSP and bipartite). Experiments have been carried out in order to evaluate both approaches using the Movielens dataset. Therefore, we consider those objectives as fulfilled.

Finally, the last objective was to compare both approaches, which has been done in the previous section, marking the last objective as completed.

In this way, it has been possible to meet all the objectives initially planned.

5.3 Future work

Finally, this section intends to briefly outline certain lines of future work:

- To study the explainability of the recommendations obtained in this kind of models, as it usually increases user satisfaction.
- In practical scenarios, users and movies involved in the system are constantly changing. Creating new models every time is not a viable option (specially in large graphs), so it would be interesting to study how this problem could be approached.
- To train a model based on GSP for all movies, in order to check its effectiveness.
- For the bipartite approach, implement different architectures to accomplish even better results.
- To apply different metrics to evaluate another aspects like diversity or coverage.

- Pre-train node embeddings by using a self-supervised task and use the obtained embeddings for the target task in order to increase the final GNN model's performance. (source).

- Pre-train node embeddings by using a self-supervised task and use the obtained embeddings with a classical machine learning algorithm or a fully connected neural network for the final downstream task. (source).

Bibliography

[1] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications, 2018. URL: https://arxiv.org/abs/1812.08434, doi:10.48550/ARXIV.1812.08434.

[2] Renjie Zhou, Samamon Khemmarat, and Lixin Gao. The impact of youtube recommendation system on video views. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 404–410, 2010. doi:10.1145/1879141.1879193.

[3] Oliver Hinz and Jochen Eckert. The impact of search and recommendation systems on sales in electronic commerce. *Business & Information Systems Engineering*, 2(2):67–77, 2010. doi:10.1007/s12599-010-0092-x.

[4] F. Maxwell Harper and Joseph A. Konstan. 2015. *The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4), 19. doi:10.1145/2827872.

[5] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 346–353, 2019. URL: https://ojs.aaai.org/index.php/AAAI/article/view/3804.

[6] Ziheng Duan, Yueyang Wang, Weihao Ye, Qilin Fan, and Xiuhua Li. Connecting latent relationships over heterogeneous attributed network for recommendation. *Applied Intelligence*, pages 1–19, 2022. URL: https://arxiv.org/abs/2103.05749.

[7] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi:10.1109/TNNLS.2020.2978386.

[8] Justin Gilmer, Samuel Schoenholz, Patrick Riley, Oriol Vinyals, and George Dahl. Neural message passing for quantum chemistry. 04 2017.

[9] Alejandro Ribeiro. *Graph Neural Networks Course*. University of Pennsylvania (Penn), 2020. URL: https://gnn.seas.upenn.edu/lectures/lecture-3/.

[10] Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro. Graphs, convolutions, and neural networks: From graph filters to graph neural networks. *IEEE Signal Processing Magazine*, 37(6):128–138, 2020. doi:10.1109/MSP.2020.3016143.

[11] Zecheng Zhang, Danni Ma, and Xiaohan Li. Link prediction with graph neural networks and knowledge extraction. URL: http://cs230.stanford.edu/projects_spring_2020/reports/38854344.pdf.

[12] Weiyu Huang, Antonio G. Marques, and Alejandro R. Ribeiro. Rating prediction via graph signal processing. *IEEE Transactions on Signal Processing*, 66(19):5066–5081, 2018. doi:10.1109/TSP.2018.2864654.

[13] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. URL: https://arxiv.org/abs/1903.02428.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017. URL: https://arxiv.org/abs/1706.02216, doi:10.48550/ARXIV.1706.02216.

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL: https://arxiv.org/abs/1710.10903, doi:10.48550/ARXIV.1710.10903.

[16] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey. *ACM Computing Surveys*, 05 2022. doi:10.1145/3535101.

[17] Sentence-transformers/all-distilroberta-v1 from hugging face. URL: https://huggingface.co/sentence-transformers/all-distilroberta-v1.