

**IMPROVEMENTS IN RECOMMENDER SYSTEMS: ADDRESSING  
DATA SPARSITY AND CLASS IMBALANCE WITH ATTRIBUTE  
INTEGRATION AND NETWORK SCIENCE**

(ÖNERİ SİSTEMLERİNDEKİ İYİLEŞTİRMELER: ÖZNİTELİK  
ENTEGRASYONU VE AĞ BİLİMİ İLE VERİ AZLIĞI VE SINIF  
DENGESİZLİĞİNİN ELE ALINMASI)

by

**ELIF ECE ERDEM, B.S.**

**Thesis**

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

**MASTER OF SCIENCE**

in

**COMPUTER ENGINEERING**

in the

**GRADUATE SCHOOL OF SCIENCE AND ENGINEERING**

of

**GALATASARAY UNIVERSITY**

**JULY 2024**

Approval of the thesis:

**IMPROVEMENTS IN RECOMMENDER SYSTEMS: ADDRESSING  
DATA SPARSITY AND CLASS IMBALANCE WITH ATTRIBUTE  
INTEGRATION AND NETWORK SCIENCE**

submitted by **ELIF ECE ERDEM** in partial fulfillment of the requirements for  
the degree of **Master of Science in Computer Engineering Department,**  
**Galatasaray University** by,

**Examining Committee Members:**

Assoc. Prof. Dr. Günce Keziban ORMAN  
Supervisor, **Computer Engineering Department**  
**Galatasaray University**

\_\_\_\_\_

Assist. Prof. Dr. Pınar ULUER  
**Computer Engineering Department**  
**Galatasaray University**

\_\_\_\_\_

Assist. Prof. Dr. Atay ÖZGÖVDE  
**Computer Engineering Department**  
**Boğaziçi University**

\_\_\_\_\_

**Date:**

\_\_\_\_\_

## ACKNOWLEDGMENTS

This thesis is supported by the bilateral project of the Scientific and Technological Research Council of Türkiye (TÜBİTAK), under grant number 122N701, with the CampusFrance, within the scope of Hubert Curien Partnerships (PHC) project number 49032VB.

I would like to express my deep gratitude to Dr. Günce Keziban ORMAN as my research supervisor, for her patient guidance, enthusiastic encouragement, and useful critiques of this research work.

In addition, I would like to thank my supportive family...

# TABLE OF CONTENTS

|                                                     |        |
|-----------------------------------------------------|--------|
| ACKNOWLEDGEMENTS . . . . .                          | iii    |
| TABLE OF CONTENTS . . . . .                         | iv     |
| LIST OF FIGURES . . . . .                           | viii   |
| LIST OF TABLES . . . . .                            | x      |
| LIST OF ALGORITHMS . . . . .                        | xi     |
| ABSTRACT . . . . .                                  | xii    |
| RÉSUMÉ . . . . .                                    | xiii   |
| ÖZET . . . . .                                      | xiv    |
| <br>1 INTRODUCTION . . . . .                        | <br>1  |
| 1.1 Overview & Motivation . . . . .                 | 1      |
| 1.2 Problem Statement . . . . .                     | 4      |
| 1.3 Contributions . . . . .                         | 7      |
| <br>2 RECOMMENDATION SYSTEMS . . . . .              | <br>10 |
| 2.1 Overview . . . . .                              | 10     |
| 2.1.1 Categories of Recommendation System . . . . . | 10     |
| 2.1.1.1 Content-Based Filtering (CBF) . . . . .     | 11     |
| 2.1.1.2 Collaborative Filtering (CF) . . . . .      | 13     |
| Memory-Based Approaches . . . . .                   | 14     |
| Model-Based Approaches . . . . .                    | 14     |
| 2.1.1.3 Hybrid Models . . . . .                     | 15     |
| Weighted . . . . .                                  | 15     |
| Mixed . . . . .                                     | 15     |

|       |                                                                    |    |
|-------|--------------------------------------------------------------------|----|
|       | Switching . . . . .                                                | 16 |
|       | Feature Combination . . . . .                                      | 16 |
|       | Feature Aggregation . . . . .                                      | 16 |
|       | Cascade . . . . .                                                  | 16 |
|       | Meta-Level . . . . .                                               | 17 |
| 2.2   | Existing Challenges . . . . .                                      | 17 |
| 2.2.1 | Data Sparsity . . . . .                                            | 17 |
| 2.2.2 | Class Imbalance . . . . .                                          | 19 |
| 2.2.3 | Cold-Start . . . . .                                               | 20 |
| 2.2.4 | Scalability . . . . .                                              | 21 |
| 2.2.5 | Diversity . . . . .                                                | 22 |
| 2.3   | Evaluation Metrics . . . . .                                       | 23 |
| 2.3.1 | Precision@K . . . . .                                              | 23 |
| 2.3.2 | Mean Average Precision@K (MAP@K) . . . . .                         | 24 |
| 2.3.3 | Recall@K . . . . .                                                 | 24 |
| 2.3.4 | Normalized Discounted Cumulative Gain (NDCG) . . . . .             | 24 |
| 2.3.5 | Hit Rate . . . . .                                                 | 25 |
| 2.3.6 | Mean Reciprocal Rank (MRR) . . . . .                               | 26 |
| 2.4   | User-based Personalized Recommendation Frameworks . . . . .        | 26 |
| 2.4.1 | Bayesian Personalized Ranking (BPR) . . . . .                      | 27 |
| 2.4.2 | Neural Graph Collaborative Filtering (NGCF) . . . . .              | 28 |
| 2.4.3 | Light Graph Convolutional Neural Network (LightGCN) . . . . .      | 31 |
| 2.4.4 | Neighborhood-based Neural Collaborative Filtering (NNCF) . . . . . | 32 |
| 2.4.5 | Self-Supervised Graph Learning for Recommendation (SGL) . . . . .  | 35 |

|         |                                                                 |    |
|---------|-----------------------------------------------------------------|----|
| 3       | OUR SOLUTION FOR DATA SPARSITY . . . . .                        | 37 |
| 3.1     | Data Granularity and Network Representation . . . . .           | 37 |
| 3.2     | Experimental Design . . . . .                                   | 40 |
| 3.2.1   | Data Augmentation . . . . .                                     | 41 |
| 3.2.2   | Level Projection . . . . .                                      | 44 |
| 3.2.3   | Recommendation . . . . .                                        | 44 |
| 3.2.4   | Degrading Recommendations and Performance Measurement . . . . . | 45 |
| 4       | OUR SOLUTION TO CLASS IMBALANCE . . . . .                       | 49 |
| 4.1     | Negative Sampling . . . . .                                     | 49 |
| 4.1.1   | Negative Sampling Evolution . . . . .                           | 51 |
| 4.1.2   | Negative Sampling Algorithms . . . . .                          | 53 |
| 4.1.2.1 | Static Negative Sampling (Static NS) . . .                      | 54 |
|         | Random Negative Sampling (RNS) . . . .                          | 54 |
|         | Popularity-based Negative Sampling (PNS) . . . .                | 55 |
| 4.1.2.2 | Hard Negative Sampling (Hard NS) . . . .                        | 55 |
|         | Dynamic Negative Sampling (DNS) . . . .                         | 56 |
|         | Mixture-based DNS . . . . .                                     | 56 |
| 4.1.2.3 | GAN-based Negative Sampling . . . . .                           | 57 |
| 4.1.2.4 | Auxiliary-based Negative Sampling . . . .                       | 57 |
| 4.1.2.5 | In-batch Negative Sampling . . . . .                            | 57 |
|         | Basic In-batch NS . . . . .                                     | 58 |
|         | Debiased In-batch NS . . . . .                                  | 58 |
|         | Hard In-batch NS . . . . .                                      | 58 |
| 4.2     | Network Science Oriented Hard Negative Sampling . . . . .       | 58 |
| 4.2.1   | Path-Length Negative Sampling (PL-NS) . . . . .                 | 59 |

|       |                                                    |     |
|-------|----------------------------------------------------|-----|
| 4.2.2 | Path-Density Negative Sampling (PD-NS) . . . . .   | 62  |
| 4.2.3 | Diffusion Distance-based Negative Sampling (DD-NS) | 64  |
| 5     | EXPERIMENTS AND RESULTS . . . . .                  | 71  |
| 5.1   | Results for Data Sparsity Solution . . . . .       | 74  |
| 5.2   | Results for Class Imbalance Solution . . . . .     | 78  |
| 5.3   | Discussion . . . . .                               | 93  |
| 6     | CONCLUSION . . . . .                               | 96  |
|       | REFERENCES . . . . .                               | 99  |
|       | BIOGRAPHICAL SKETCH . . . . .                      | 112 |

## LIST OF FIGURES

|                                                                                                                                                                                                                                                                                                                                                    |    |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Figure 2.1 Process of general recommendation models (Ko et al., 2022) . . . . .                                                                                                                                                                                                                                                                    | 12 |
| Figure 2.2 Hybrid models approaches (Fayyaz et al., 2020) . . . . .                                                                                                                                                                                                                                                                                | 15 |
| Figure 2.3 A representation of the high-order connectivity and user-item interaction graph. Image courtesy from (Wang et al., 2019) . . . . .                                                                                                                                                                                                      | 29 |
| Figure 3.1 Results for “bag” search on Amazon.com . . . . .                                                                                                                                                                                                                                                                                        | 38 |
| Figure 3.2 Data Augmentation : We progressively enrich the representations of items by incorporating additional attributes into their node definitions at each level. . . . .                                                                                                                                                                      | 42 |
| Figure 3.3 Level Projection . . . . .                                                                                                                                                                                                                                                                                                              | 45 |
| Figure 3.4 Degrading Recommendations . . . . .                                                                                                                                                                                                                                                                                                     | 47 |
| Figure 3.5 Data Granularity Integration Flow Chart . . . . .                                                                                                                                                                                                                                                                                       | 48 |
| Figure 4.1 Path-Length Negative Sampling Flow Chart . . . . .                                                                                                                                                                                                                                                                                      | 68 |
| Figure 4.2 Path-Density Negative Sampling Flow Chart . . . . .                                                                                                                                                                                                                                                                                     | 69 |
| Figure 4.3 Diffusion Distance Negative Sampling Flow Chart (The images in the diffusion distance calculation process are courtesy from : <a href="https://gbertagnolli.github.io/diffudist/articles/diffusion-distances.html#references-1">https://gbertagnolli.github.io/diffudist/articles/diffusion-distances.html#references-1</a> ) . . . . . | 70 |
| Figure 5.1 Path-Length NS Approaches Results . . . . .                                                                                                                                                                                                                                                                                             | 87 |
| Figure 5.2 Path-Length NS Approaches Results . . . . .                                                                                                                                                                                                                                                                                             | 88 |

Figure 5.3 Diffusion Distance-based NS Approaches Results . . . . . 89

Figure 5.4 RNS + Diffusion Distance-based NS Approaches Results . . . . . 90

Figure 5.5 The best results from each proposed negative sampling category . . 92



## LIST OF TABLES

|                                                                                  |    |
|----------------------------------------------------------------------------------|----|
| Table 3.1 Sample data used for Level 1 . . . . .                                 | 43 |
| Table 3.2 Sample data used for Level 2 . . . . .                                 | 43 |
| Table 3.3 Sample data used for Level 3 . . . . .                                 | 43 |
| Table 3.4 Sample data used for Level 4 . . . . .                                 | 44 |
| Table 5.1 Data Statistics of Last.fm dataset . . . . .                           | 71 |
| Table 5.2 Data Sparsity of created data granularity levels . . . . .             | 75 |
| Table 5.3 Results for NGCF in Last.fm dataset for each data granularity . . . .  | 75 |
| Table 5.4 Results for LighGCN in Last.fm dataset for each data granularity . .   | 76 |
| Table 5.5 Results for SGL in Last.fm dataset for each data granularity . . . . . | 77 |
| Table 5.6 Path-Length NS Approach Results in Last.fm dataset . . . . .           | 80 |
| Table 5.7 Path-Density NS Approach Results in Last.fm dataset . . . . .          | 81 |
| Table 5.8 DD-NS Results in Last.fm dataset . . . . .                             | 83 |
| Table 5.9 RNS + DD-NS Results in Last.fm dataset . . . . .                       | 85 |
| Table 5.10 The best NS results in Last.fm dataset . . . . .                      | 91 |

LIST OF ALGORITHMS

Algorithm 1    Single Source Shortest Path Length . . . . . 61

Algorithm 2    All Pairs Shortest Path Length . . . . . 62

Algorithm 3    Estimate Paths . . . . . 63



## ABSTRACT

Recommendation systems suffer greatly from data sparsity and class imbalance, which reduces the quality of user recommendations and limits the system’s capacity to use the entire item inventory. Addressing these issues is crucial for improving recommendation accuracy and delivering well-suited suggestions to users. This thesis introduces novel ways to improve recommendation accuracy by incorporating item attributes into user-item interaction and using network science in negative sampling in collaborative filtering (CF) algorithms. Firstly, we proposed a data granularity framework that systematically integrates item attributes at different levels of detail within the bipartite network structure. This enriches the item nodes with detailed information, hence enabling the CF algorithms to potentially minimize the negative effects of data sparsity on the recommendations.

Secondly, we designed network-oriented negative sampling strategies that leverage user-item interaction structure while selecting more informative negative samples for the CF algorithm’s learning task. Unlike traditional negative sampling, these strategies personalize the selection process for each user and prioritize negative items according to different network-related factors like node distance and link density.

For the two problems mentioned, i.e. data sparsity and class imbalance, we tested our proposed solutions, i.e. the data granularity framework and network-oriented negative sampling, respectively, with large sets of experiments. The results show that our approaches mitigate the negative effects of the data sparsity and class imbalance, leading to more robust recommendations. These findings may pave the way for new developments in this field.

**Keywords :** Recommendation Systems, Graph Neural Networks, Network Science, Collaborative Filtering, Negative Sampling.

## RÉSUMÉ

Les systèmes de recommandation souffrent de la rareté des données et du déséquilibre des classes, ce qui réduit la qualité des recommandations et limite l'utilisation de l'inventaire des objets. Aborder ces problèmes est crucial pour améliorer la précision des recommandations et fournir des suggestions adaptées. Cette thèse propose de nouvelles améliorations pour les algorithmes de filtrage collaboratif (FC) en intégrant les attributs des objets dans l'interaction utilisateur-objet et en utilisant la science des réseaux pour l'échantillonnage négatif.

Nous avons d'abord proposé un cadre de granularité des données intégrant les attributs des objets à différents niveaux de détail dans l'algorithme de FC. En construisant des réseaux bipartites où les attributs des objets sont intégrés dans les nœuds, nous enrichissons les nœuds avec des informations détaillées. Cela permet aux algorithmes de FC de minimiser les effets négatifs de la rareté des données.

Ensuite, nous avons conçu des stratégies d'échantillonnage négatif orientées réseaux, exploitant la structure bipartite des interactions utilisateur-objet pour sélectionner des échantillons négatifs plus informatifs. Contrairement aux méthodes traditionnelles, ces stratégies personnalisent la sélection pour chaque utilisateur et priorisent les objets négatifs selon des facteurs comme la distance des nœuds et la densité des liens, rendant les échantillons plus pertinents.

Nous avons intégré nos méthodes d'échantillonnage négatif avec les méthodes de FC basées sur les réseaux de convolution de graphes (GCN). Les résultats montrent que nos approches atténuent les effets de la rareté des données et du déséquilibre des classes, conduisant à des recommandations plus robustes.

**Mots Clés :** Systèmes de recommandation, réseaux de neurones graphiques, science des réseaux, filtrage collaboratif, Échantillonnage négatif.

## ÖZET

Tavsiye sistemleri, veri kıtlığı ve sınıf dengesizliğinden etkilenir, bu da kullanıcı tavsiyelerinin kalitesini düşürür ve sistemin tüm öge envanterini kullanma kapasitesini sınırlar. Bu sorunların giderilmesi, tavsiye doğruluğunu artırmak ve kullanıcılara uygun öneriler sunmak için kritiktir. Bu tez, işbirlikçi filtreleme (CF) algoritmalarını iyileştirmek için öge özniteliklerini kullanıcı-öge etkileşimine dahil ederek ve negatif örnekleme için ağ bilimini kullanarak yenilikçi yollar tanıtmaktadır.

Öncelikle, CF algoritması içinde öge özniteliklerini farklı detay seviyelerinde entegre eden bir veri granülerlik çerçevesi önerdik. Öğelerin özniteliklerinin doğrudan öge düğümlerine dahil edildiği iki taraflı ağlar oluşturuyoruz. Bu, öge düğümlerini detaylı bilgilerle zenginleştirir ve CF algoritmalarının veri kıtlığının olumsuz etkilerini en aza indirmesini sağlar.

İkinci olarak, kullanıcı-öge etkileşimlerinden oluşturulan iki taraflı ağ yapısını kullanarak daha bilgilendirici negatif örnekler seçerken ağ yönelimli negatif örnekleme stratejileri tasarladık. Geleneksel yöntemlerin aksine, bu stratejiler her kullanıcı için seçim sürecini kişiselleştirir ve negatif öğeleri düğüm mesafesi ve bağlantı yoğunluğu gibi faktörlere göre önceliklendirir. Bu yaklaşımlar, negatif örneklerin daha alakalı ve öneri modelini geliştirmede daha etkili olmasını sağlar.

Negatif örnekleme yöntemlerimizi Grafik Konvolüsyonel Ağlar (GCN) tabanlı CF yöntemleriyle entegre ettik ve veri granülerliği üzerinde deneyler yaptık. Sonuçlar, yaklaşımlarımızın veri kıtlığı ve sınıf dengesizliğinin olumsuz etkilerini hafiflettiğini ve daha sağlam önerilere yol açtığını göstermektedir. Bu bulgular, bu alanda yeni gelişmelere zemin hazırlayabilir.

**Anahtar Kelimeler :** Tavsiye Sistemleri, Grafiksel Sinir Ağları, Ağ Bilimi, İşbirlikçi Filtreleme, Negatif Örnekleme.

# 1 INTRODUCTION

## 1.1 Overview & Motivation

Recommendation systems play a crucial role in assisting users with decision-making across a variety of applications, from e-commerce to entertainment services (Bobadilla et al., 2013). Their performance, however, lies in the quality and quantity of the data available about the user and their interaction with the studied system. In recent years, collaborative filtering (CF) algorithms that naturally depend on user-item interactions, have gained popularity due to their significant performance in recommender systems (He et al., 2020; Wang et al., 2019; Wu et al., 2021; Bai et al., 2017). These approaches utilize explicit interactions, such as ratings or likes/dislikes, which reveal user preferences directly, or implicit interactions, like purchases or clicks, which indicate interest but not explicit preference (Ricci et al., 2010). Recommendation systems intend to enhance the user experience by suggesting items that users are likely to appreciate or buy. There are some major challenges that negatively affect the success of recommender systems. Our goal in this thesis is to address two of these challenges to improve the accuracy of such recommendations based on user-item interactions.

First, recommendation systems typically contain extensive item collections, but users interact with just a small subset, leading to sparse interaction data. This *data sparsity* presents a major challenge, as insufficient user-item interactions hinder the system's effectiveness, which in turn manifests as poor user experience (Natarajan et al., 2020). Imagine a music streaming service in which you would like to learn about new artists that fit your taste in music. However, you might receive recommendations for popular, generic artists, or repeatedly get suggestions for the same artist with different songs that you have already heard multiple times. This is a typical example of issues caused by data sparsity. Similar to how it impacts music recommendation, this problem compromises e-commerce products, movies, hotels, and other systems' capacities to provide a variety of tailored user experiences. As remarked, data sparsity poses a significant obstacle for recommendation systems. Due to insufficient user-item interaction data,

recommendation algorithms struggle to capture user preferences with precision. This leads to recommendations that are either generic and of no value or irrelevant to the user’s actual interests.

Second, *class imbalance*, which is essentially an analytical difficulty, greatly degrades the performance of recommendation systems. This issue often arises from data sparsity, where insufficient user interactions lead to an imbalance in the data. In a vicious circle, as data sparsity persists, the problem deepens, further complicating accurate prediction. Specifically, in the context of recommendation systems, classes are defined based on user-item interactions : interactions such as clicks, purchases, or ratings are classified as positive, a.k.a 1, indicating user interest. Conversely, the absence of interaction is classified as negative, a.k.a 0 (Ricci et al., 2010). This classification is straightforward, yet it becomes problematic in conditions of data sparsity (Natarajan et al., 2020). For instance, imagine an online bookstore where most customers only browse a few popular novels without buying or rating them, resulting in vast amounts of class 0, i.e. non-interactions, and very few class 1 i.e. interactions. This class imbalance poses a significant algorithmic problem, undermining accurately learning genuine user preferences.

Although data sparsity and class imbalance seem to be semantically similar problems, data sparsity describes the incompleteness of the data, while class imbalance is the problem of label imbalance that later methods will face. More precisely, data sparsity is a problem in the modeling of the data, while class imbalance is a problem in the algorithms that will be used for recommendation. We address these problems in separate discussions in order to highlight the distinct effects they have on the efficiency of recommendations. By addressing these challenges under two distinct headings, we aim to outline specific strategies that mitigate their negative impacts and enhance the overall accuracy of recommendation systems.

In an attempt to alleviate the challenge of data sparsity, researchers have looked into supplementing the recommendation process with data other than interactions, like user attributes (like age, gender, location, and hobbies) and item attributes (like type, material, and descriptions). By using these attributes, their purpose is to provide a richer context for understanding user preferences and item characteristics (Tahmasebi et al., 2021; Volkovs et al., 2017b; Safoury and Salah, 2013). Content-based filtering (CBF)

algorithms take advantage of this additional information. They create user profiles based on the items’ attributes with which users have previously interacted (Wang and Wang, 2014). This can include item category, item brand, item purchased time, and so on. By combining these attributes, CBF algorithms can recommend similar items by identifying user and/or item segments with similar tastes, even if those items have not received much user interaction.

However, traditional CBF algorithms, despite their utility, primarily focus on item content and often fail to integrate user-item interactions effectively, which are crucial for generating accurate recommendations in dynamic environments. In recent years, CF algorithms that utilize Graph Convolutional Networks (GCNs) have demonstrated significantly better results compared to traditional CBF approaches (He et al., 2020; Wu et al., 2021; Wu, Sun, Zhang, Xie and Cui, 2022). GCNs excel in capturing complex and non-linear relationships by leveraging the graph structure, which effectively encodes user and item interactions. This methodology not only conforms to the inherent network nature of user-item interactions but also enhances recommendation quality by utilizing deep learning to infer user preferences from sparse data. Given these advantages, this thesis focuses on GCN-based CF algorithms instead of traditional CBF methods, recognizing the latter’s limitations.

Notably, despite its outperformance over traditional techniques, there is much evidence showing that a lack of sufficient user-item interactions still proves to be an obstacle for GCN-based CF algorithms (Wu, Sun, Zhang, Xie and Cui, 2022). To address this, we propose to incorporate item content into collaborative representations, thus reducing the negative impact of data sparsity on GCNs. By integrating item content into the graph structure, we enhance the GCN’s ability to analyze and learn from both the explicit attributes of items and the implicit patterns derived from user interactions. This method enables us to utilize content data to refine the underlying graph representation, ensuring that our recommendations are both accurate and personalized, effectively addressing the complexities of real-world data sparsity in recommendation systems.

Besides, class imbalance, which is a secondary consequence of data sparsity, is often effectively tackled using negative sampling strategies in the GCN solutions (Yang et al., 2020). This method addresses sparsity by selecting “negative examples”, which are items the user is unlikely to be interested in from a substantially larger pool of negatives

during the training (Yang et al., 2024). During training, the CF algorithms compare examples that are positive and negative in order to learn how to distinguish between relevant and irrelevant data. This distinction is achieved by minimizing a loss function that penalizes the model for incorrectly predicting user preferences. Consequently, even with sparse data, the model can become more adept at understanding user preferences and making pertinent item recommendations. At this point, since the algorithm’s recommendations are very dependent on the positive and negative instances it receives as input, the choice of which negative items to use is an important factor affecting the outcome of the algorithm. Traditionally, negative sampling methods often rely on uniform random sampling from a set of all items except items the user has currently interacted with. However, they can be inefficient due to the vast number of irrelevant items they sample from (Yang et al., 2024).

Our research is motivated by these issues, and it aims to alleviate data sparsity and class imbalance limits in CF.

## 1.2 Problem Statement

We specifically focused on the following research problem, which we can formally express as follows : “How can we leverage item attributes and/or network topology information to build a powerful recommender framework that effectively reduces the limitations of data sparsity and class imbalance to have improved recommendation accuracy ?”

To address this pervasive issue, we investigate data sparsity and class imbalance from two angles in this thesis :

### 1. Item Representation with Attributes :

In this thesis, we focus on the data representation granularity level and its potential impact on refining and enhancing the success of final recommendations. This issue, which we refer to as the *data granularity problem*, involves figuring out the best level of detail to use when incorporating item attributes into the input graph for CF algorithms. More precisely, the data granularity problem includes choosing between different levels of detail for item attributes, ranging from basic

descriptions to detailed specifics about materials or styles. Our approach enriches the input data by adding various item properties at different levels of granularity.

To illustrate, consider the example of product attributes on the Amazon website. A simple product description such as “T-Shirt” gives some basic categorization. A bit more detail would be the material, for example, “cotton T-shirt”, and a bit more refinement could detail the style, like “cotton T-shirt with long sleeves and a red button-down”. By integrating such levels of detail inside the input graph, we can create several representations for the same items while assisting with different granularity levels. For instance, one representation might only use the broad category (T-shirt), another might include material (cotton T-shirt), and yet another might incorporate all available details (cotton T-shirt with long sleeves and red button-down). Recommending cotton T-shirts instead of T-shirts may increase success. However, adding more detail at the level of a long-sleeved, red button-down T-shirt may not facilitate increased success. In data representation, a dedicated methodology is required to decide which level ideally represents data and gives a more accurate recommendation. There is a gap in the literature on determining the ideal granularity of data. Our focus is to increase the overall quality of suggestions by revealing hidden relationships between items in the input graph by including these different levels of information. The challenge lies in selecting the most appropriate data granularity level for enhancing CF accuracy.

## 2. Network-Oriented Negative Sampling

Negative sampling is a useful method for addressing the class imbalance problem in CF algorithms by choosing negative items that are unlikely to be of interest to users during the learning phase. A widely used approach is uniform random negative sampling, which involves choosing negative items without any prerequisite. However, the probability of picking the negatives that will be the most useful for learning is very low, since the the negative labeled interactions amount is huge. therefore uniform random selection may not be effective for recommendation (Yang et al., 2024). Additionally, popularity-based sampling tends to ignore potentially relevant less popular items in favor of popular ones because it is dependent on item popularity, which may bias the model toward popular items. This can result in a lack of variety in recommendations, as the model promotes

items that are generally popular over ones that may be suited for a specific user (Mikolov, Sutskever, Chen, Corrado and Dean, 2013).

When dealing with class imbalance, selecting informative negative examples becomes even more critical. This focuses the model’s learning on relevant negative examples that are more likely to improve its ability to distinguish between user preferences and non-preferences. In response to these shortcomings, this thesis introduces a new concept called network-science-oriented negative sampling. Traditional uniform random and popularity-based sampling miss the fine-grained relation between users and items, which results in a loosening effect on training. We propose to address this by deriving structural features of the network representing the user-item interactions, where each user is connected with items they have shown positive interest in.

For example, consider a user who buys a “blue T-shirt” through the Amazon website. Traditional negative sampling methods might select a “kitchen appliance” as a negative sample, which is likely to be irrelevant and uninformative. In contrast, our approaches will examine items closer to or further from the user’s known preferences within the network. An item closer to a “blue T-shirt” might be a “pink T-shirt”. While this is an item outside the user’s preference, it resembles a positive item but is, in fact, negative. Conversely, an item further from the blue T-shirt could be “yellow pants”, which is more distantly related to the user’s clothing preference. By considering the proximity of items within the network, we can select negative samples that are more contextually relevant and informative for the model’s learning process.

Therefore, this thesis aims to identify both negative examples that resemble positive items and those that do not, in order to enhance the model’s effectiveness in environments with class imbalances. We aim to fill a gap in the literature by proposing a series of new approaches based on network science to identify the actual negative examples and thus mitigate the negative effects of class imbalance.

### 1.3 Contributions

The challenge of data sparsity and class imbalance becomes paramount concerning the design of effective recommender systems. We propose two distinct novel approaches to improve recommendation accuracy by mitigating these limitations :

- **Data Granularity Framework** : The novel data granularity framework explores the impact of using item attributes in the bipartite graph representation, which is used as input for CF algorithms. The framework systematically resolves the negative effects of the data sparsity issue.

This approach contributes to the field by :

- Designing a systematic methodology of data granularity framework for optimizing bipartite network representations using item attributes, thereby improving recommendation accuracy tested through CF algorithms.
  - Introducing an objective performance evaluation system within the framework design, so that the achievements of different levels of data granularity can be compared fairly with each other.
  - Proposing an exhaustive experimental design that explores all possible attribute combinations to determine the effectiveness of the proposed framework.
  - Demonstrating and validating the efficiency of the proposed framework by using 3 state-of-the-art GCN-based algorithms, NGCF, LightGCN, SGL (Wang et al., 2019; He et al., 2020; Wu et al., 2021) and 6 different performance evaluation metrics, Recall@K, Precision@K, NGCF, Hit Rate, MRR, MAP, on the well-known music dataset.
- 
- **Network-Oriented Negative Sampling** By leveraging the network structure of user-item interactions, we propose a set of network-science-oriented negative sampling techniques designed to address the class imbalance problem. We select negative items by analyzing “how far” the items are from the position of each user on the bipartite network. We determine this distance using various methods and report their performances based on the experiments conducted in this thesis. These techniques aim to overcome the limitations of traditional negative sampling methods, such as biased learning and poor performance in negative classes.

Our network-science-oriented methods try to sample negative items from specific network positions surrounding the target user.

This work contributes to the field by :

- Introducing 10 novel negative sampling strategies that utilize different aspects of network topology to mitigate the inefficiencies inherent in uniform random sampling in class-imbalanced data environments.
- Experimentally representing our proposed negative sampling strategies effectiveness on a well-known music dataset with LightGCN (He et al., 2020) using 3 different metrics, Precision@K, Recall@K, and NDCG, respectively.
- Enhancing CF model training by early convergence with focusing on more informative negative cases, thereby improving model performance and recommendation effectiveness in class-imbalanced settings.

Such two approaches, put forward in isolation by the thesis, are thereby immensely valuable as contributions to the construction of robust recommendation engines that can handle both data sparsity and class imbalance in getting more accurate recommendations.

In this regard, the subsequent sections of this thesis are organized in the following order :

Chapter 2 is a literature review of recommender systems; their various categories, existing limitations, typical evaluation metrics, and user-based recommendation frameworks that explore several methods such as Neural Graph Collaborative Filtering (NGCF), Light Graph Convolutional Neural Network (LightGCN), and Self-Supervised Graph Learning for Recommendation (SGL).

In chapter 3, we propose a novel experimental design, leveraging both data granularity and network representation in order to address data sparsity. Later in Chapter 4, we introduce the class imbalance problem that results from the uneven distributions of positive and negative interactions in recommender systems. In this regard, we discussed several negative sampling algorithms like static, hard negative sampling, and

the GAN-based approaches in the literature. We present our novel, network-science-oriented negative sampling approaches here.

The results obtained after running our solutions for data sparsity and class imbalance were presented in Chapter 5, with the final section of this chapter discussing the implications of these results.



## 2 RECOMMENDATION SYSTEMS

### 2.1 Overview

The internet's explosive growth in the last several decades has produced an abundance of information that overwhelms consumers with a wide variety of readily available content (Beel et al., 2013). The task of efficiently sorting and retrieving pertinent information from a large amount of available data is made more difficult by this abundance. For example, millions of products and personalized recommendation systems are often found on one platform on e-commerce websites. Despite the convenience of these platforms, consumers may experience information overload due to the abundance of options accessible.

One possible solution for information overload is recommender systems. Based on users' prior interactions and preferences, these systems employ algorithms to forecast potential user interests (Adomavicius and Tuzhilin, 2005). These systems have attracted a lot of attention from researchers recently since they have such a practical application value. The key objective here is attaining an estimate for a given user's preference level when it comes to any particular item  $i$  that belongs in set  $I$ . This is achieved through utilizing learned representations centered around both the user and the desired item. We can formulate this purpose as follows. Here,  $y_{u,i}$  can be considered as the  $u$  user's score of preferring the  $i$  item.

$$y_{u,i} = f(h_u^*, h_i^*) \quad (2.1)$$

where  $h_u^*$  and  $h_i^*$  are ultimate representation of user  $u$  and item  $i$ , respectively. Additionally,  $f(\cdot)$  returns a numeric value for the probability that user  $u$  prefers  $i$ .

#### 2.1.1 Categories of Recommendation System

Recommendation systems present many classification schemes, which makes it easier to understand their varied methods. One way to do this classification is to look into how they collect user preference data. We can examine the recommendation algorithms

based on the data they used under two main categories : Explicit feedback and implicit feedback.

Users can share their preferences directly and openly through comments, likes/dislikes, or simple ranking systems. This approach has the advantage of openness and directness as users provide specific information about their likes and dislikes. However, this tactic requires active user participation, which is difficult to achieve in real-world data, and may not fully capture the subtleties of users' opinions about items, since the feedback is often given on a numerical scale (e.g., a range from 1 to 5) or as binary choices (e.g., like or dislike) (Ricci et al., 2010).

Implicit feedback, on the other hand, infers preferences from user behavior. A relevancy score is assigned to particular user actions with an item, such as buying, bookmarking, dismissing, or saving. The primary advantage of implicit feedback is that it does not require direct user engagement, potentially capturing more genuine user behavior. However, implicit feedback can contain biases (Ricci et al., 2010). For instance, while shopping on an e-commerce platform, the user may accidentally click on a product they do not want to buy. If the recommendation system only tracks the user's clicks and offers item recommendations based on this data, it may recommend products that the user may not actually like due to some incorrect clicks.

Furthermore, to improve recommendation quality, some recommendation systems may include additional data sources, like social network connections or demographic data (Wu, Sun, Zhang, Xie and Cui, 2022).

Finally, recommendation problems can be addressed using different techniques, which fall into three major categories : Collaborative Filtering, Content-Based Filtering, and Hybrid methods (Bobadilla et al., 2013).

The general recommendation models' process can be illustrated in 2.1.

#### **2.1.1.1 Content-Based Filtering (CBF)**

CBF is a method that recommends items with similar characteristics to those that a consumer has previously chosen. The intrinsic qualities of the items themselves are

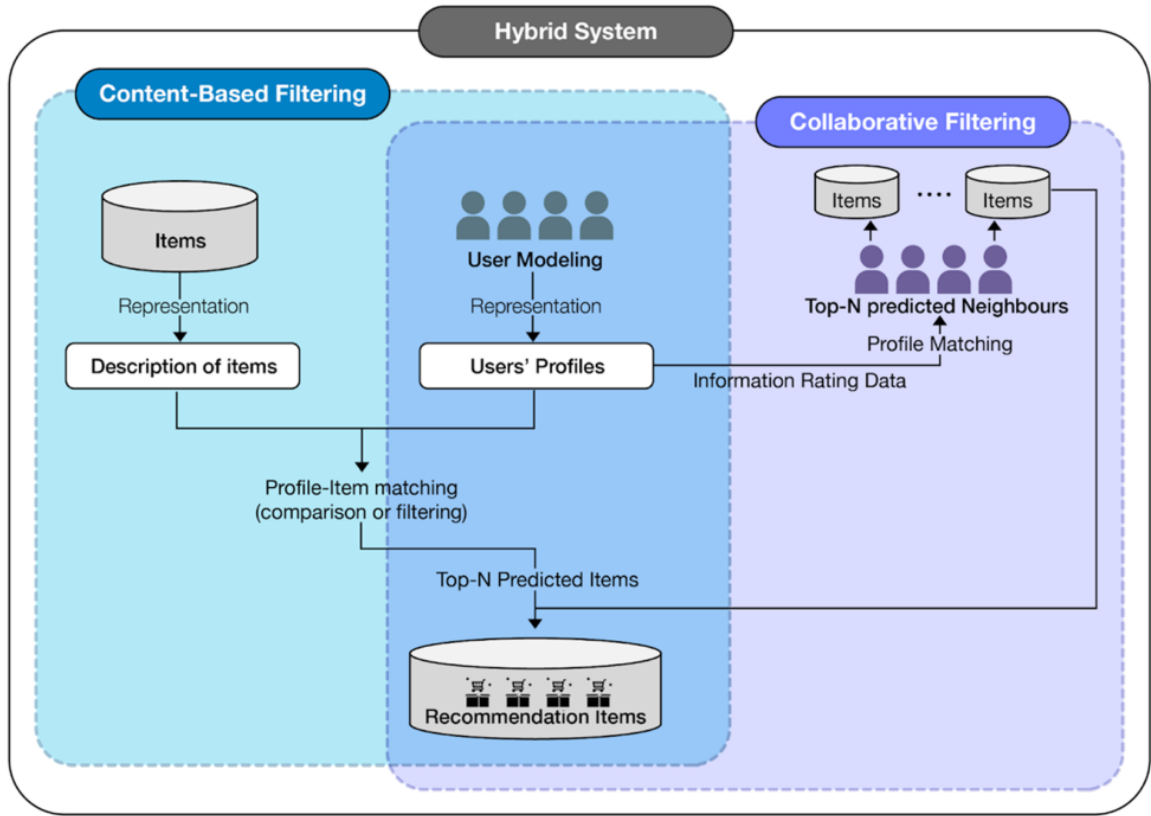


FIGURE 2.1 – Process of general recommendation models (Ko et al., 2022)

what motivates this recommendation method (Wang and Wang, 2014). In its simplest form, the algorithm finds items whose attributes are similar to those the user has previously chosen. Early recommendation systems relied heavily on CBF, which was used as the foundational model for more complex recommender system architectures.

CBF recommendation systems offer several advantages. Firstly, they excel at personalization by leveraging a user’s ratings and preferences to suggest items. This user-centric approach ensures recommendations are tailored to individual tastes. Secondly, these systems boast transparency, allowing users to understand why specific items are recommended by highlighting relevant features that match their profile. Finally, CBF recommenders excel at discovering and suggesting new items. They can recommend even unrated content that matches a user’s interests, helping to address the issue where new products struggle to gain attention until a significant number of people have interacted with them (Ricci et al., 2010).

However, CBF techniques face challenges in capturing the full spectrum of user preferences since they often rely on text mining approaches like semantic analysis (Di Noia

et al., 2012), Term-Frequency Inverse Document Frequency (TF-IDF) (Chiny et al., 2022), or classification algorithms like Naive Bayes (Shuxian and Sen, 2019). These methods can be limited by the amount and type of information available. While these methods excel at identifying patterns within a user’s past selections, they struggle to incorporate broader community preferences. This can lead to a narrowed recommendation scope, where items containing specific keywords a user hasn’t interacted with before are never suggested. This lack of consideration for user communities further limits the diversity of the recommended items. For instance, someone who enjoys rock music might not be recommended jazz because they have not interacted with that genre yet. Furthermore, new users with limited rating history encounter difficulties, as the system requires sufficient data to build an accurate user profile and deliver effective recommendations (Ricci et al., 2010).

#### **2.1.1.2 Collaborative Filtering (CF)**

Using user collective intelligence, CF has become the most popular recommender system technique (Wang et al., 2015). To anticipate user preferences, CF gives precedence to previous user-item interactions, in contrast to CBF approaches that concentrate on user and/or item features (Chen, Hua, Chang, Wang, Zhang and Kong, 2018). These interactions can involve relationships between users with similar preferences or relationships between items that are frequently rated together (Beel et al., 2015).

Essentially, CF creates a user-item matrix to document users’ preferences for different items. This matrix is a table where rows represent users and columns represent items, with each cell containing a value that indicates the user’s preference for a particular item (e.g., a rating, a purchase, or another form of interaction). By analyzing this matrix, the algorithm can identify users with similar preferences and recommend items that these users rate highly. These like-minded users form a “neighborhood,” and the assumption is that items liked by the user’s neighbors will also suit the user. The results of CF can either be a list of suggested items that are most likely to be liked by the user or predicted rating scores for specific items.

CF techniques offer several advantages over CBF methods. Since CF relies on the collective experiences and ratings of users, it excels at identifying hidden relationships and

patterns that are not immediately obvious. This enables CF to suggest items to users that they might find interesting, even if these items don't fit their usual preferences, leading to unexpected and delightful discoveries.

However, CF techniques have several limitations : One of the most important limitations is the (Kunaver and Požrl, 2017) cold start problem, which occurs when the system does not have enough information about a new person or item to make appropriate recommendations. New customers may sign up without having rated enough items, or new items may be added that have not been rated at all. Because CF depends on user interaction data, it cannot provide appropriate recommendations if there is insufficient data. One more difficulty is data sparsity, which is the state in which there are few user ratings, leading to a sparse user-item matrix (Kotkov et al., 2016). Users who rate a small set of all items may experience this. A sparse matrix makes it difficult to find similar users or items, ultimately leading to less effective recommendations. Finally, as the amount of data increases, CF systems may struggle to scale effectively.

CF approaches use model-based or memory-based ways to analyze these relationships and produce recommendations (Sharma et al., 2017).

**Memory-Based Approaches** Memory-based CF, also known as neighborhood-based or user-item filtering uses previous interactions to locate similar users or items and uses that information to anticipate user preferences. Then, suggestions are developed based on these similarities. For example, for a target item that the user has not yet rated, a weighted average of ratings from comparable individuals or items can be calculated. User-based methods recommend items liked by users with similar tastes. Item-based approaches suggest items that are comparable to items the user has already interacted with. This is relatively simple and works well with small amounts of data, but may not be as effective when dealing with large user bases and changing preferences (Ekstrand et al., 2011).

**Model-Based Approaches** Model-based CF learns associations between users, items, and interactions by employing machine learning or data mining techniques like matrix completion (Koren et al., 2009), clustering (Abbasi-Moud et al., 2021), and dimensionality reduction (e.g. SVD (Sarwar et al., 2002)) to develop a statistical model. This

model can suggest previously unseen items and find hidden patterns. Additionally, auxiliary data other than user interactions can be fed to the models, and these models can scale effectively with large data sets. However, implementation is more difficult and the quality of training data is critical (Pujahari and Sisodia, 2020).

### 2.1.1.3 Hybrid Models

Hybrid approaches merge both CF and CBF approaches to employ their complementary strengths together with enhancing recommendation accuracy (Adomavicius and Tuzhilin, 2005). In Figure 2.2, one can find some of the combination approaches for hybrid models.

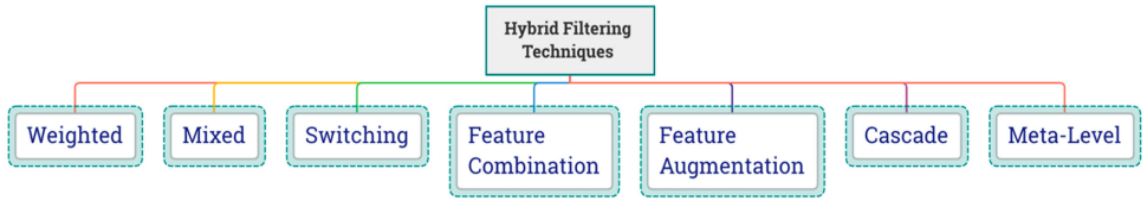


FIGURE 2.2 – Hybrid models approaches (Fayyaz et al., 2020)

**Weighted** A weighted-hybrid recommender system gives each algorithm a weight, based on how well each algorithm has been predicted in the past. In this way, the system prioritizes strategies that are effective in recommending items that users find attractive with a higher weight.

However, there is a drawback to this approach. It assumes that every recommendation technique works equally well for every item, which isn't always the case. Some methods may better recommend certain types of items (Kardan and Ebrahimi, 2013).

**Mixed** The mixed recommendation system is effective in generating a large number of recommendations at the same time. Unlike weighted hybrid approaches, it is not intended to combine data from multiple recommenders into a single score. Instead, it displays each recommender's suggestions side by side in a long list. This could be handy for users who want to view multiple alternatives (Fayyaz et al., 2020).

**Switching** A switching recommender system determines the appropriate technique based on the user’s situation. After analyzing user profiles, it chooses the recommender (e.g., CF or CBF) with the highest likelihood of delivering accurate predictions (Fayyaz et al., 2020).

**Feature Combination** Feature Combination combines the features identified by one recommendation algorithm and utilizes them in another recommendation algorithm. This is especially helpful when combining CBF (looking at item features) with CF (looking at what other users liked).

Through the treatment of CF data as additional features per item, a CBF can take user preferences into account, independent of an item’s rating count. This reduces the impact of the “cold start” issue (Fayyaz et al., 2020)

**Feature Aggregation** Utilizing the reasoning of another recommender system, the feature aggregation technique enhances recommendations by appending more attributes (data points). It is especially helpful if we wish to add more information or data sources to our robust core recommendation technique. The working mechanism is as follows : Firstly, the initial recommender system forecasts an item’s rating. Then, a new feature for that item is created using this expected rating. Finally, in order to arrive at a final recommendation, the second recommender system combines its own features with this new one (Kardan and Ebrahimi, 2013).

**Cascade** The cascade method prioritizes stronger recommendation methods over weaker ones, ensuring that the final recommendations come from the most reliable source. Initial recommendations are generated by a high-priority recommender, which assesses items and provides scores. If several items receive similar scores (resulting in a “tie”), the system then employs a lower-priority recommender to break the tie. This secondary recommender offers additional insights to differentiate between the closely scored items. The lower-priority recommender is only used in cases of ties. It does not influence the recommendations for items that have already been clearly rated as either good or bad by the high-priority recommender. This approach maintains the integrity of the stronger method’s recommendations and provides finer granularity only when

necessary (Kardan and Ebrahimi, 2013).

**Meta-Level** Meta-level approaches involve using a learned model from one recommender to improve another recommender. Unlike feature augmentation, which enhances the input features of a recommender system, meta-level techniques utilize the entire learned model to inform the recommendations. This approach provides a comprehensive representation of the user’s preferences, thereby simplifying and improving collaborative filtering methods (Kardan and Ebrahimi, 2013).

## 2.2 Existing Challenges

In the recommender systems domain, data is generally arranged into a user-item matrix. The rows in this matrix are users, while the columns are items. Each matrix element consists of a collection of activities that a given user performs on a specific item. A matrix element is usually a single number that represents a rating that is stated explicitly or that is deduced from the user’s behavior (Grčar et al., 2006).

However, working with this matrix structure presents several challenges due to the inherent properties of the data. Recommendation systems suffer from several challenges due to the properties of the data, including data sparsity, class imbalance, cold start, scalability, and diversity. We will attempt to go into further depth about these difficulties in this subsection.

### 2.2.1 Data Sparsity

Most large-scale recommendation applications have a huge number of users and items. As users usually interact with a small fraction of the available items, user-item interaction data in recommendation systems is often sparse. Consequently, the vast majority (average up to 99%) of these matrix entries are 0 (Grčar et al., 2006). This high level of sparsity means that it is common for comparisons between any two users regarding a specific item to yield no interactions (both entries being 0) (Massa and Bhattacharjee, 2004).

The sparsity problem —a crucial obstacle in CF— is introduced by the sparse matrix’s manifestation. Since similarity measures are the foundation of CF, meaningful comparisons across sparse vectors require overlapping values. High degrees of sparsity make it more difficult for trustworthy neighborhoods to form, which reduces the efficiency of CF. Furthermore, a large number of unrated items and inactive users may be present in highly sparse data, which would limit their ability to participate in the CF process (Grčar et al., 2006).

The sparsity ( $s$ ) can be mathematically represented as the ratio of observed interactions to all conceivable interactions, as given by Eq 2.2.

$$s = \frac{\text{Number of observed interactions}}{\text{Total number of possible interactions}} \quad (2.2)$$

This can also be written in terms of the user-item interaction matrix in Eq 2.3 :

$$s = \frac{\sum_{u=1}^m \sum_{i=1}^n I(r_{ui} \text{ is observed})}{m \times n} \quad (2.3)$$

where the user-item interaction matrix is denoted by  $R$ .  $m$  is the number of users and  $n$  is the number of items. The components of  $R$ , indicated as  $r_{ui}$ , symbolize the user  $u$ ’s interaction (e.g., rating, click, purchase history) with item  $i$ . The indicator function  $I(\cdot)$ , returns 1 in the case that the condition within is true and 0 otherwise (Natarajan et al., 2020).

It is difficult to appropriately represent user preferences and provide intelligent suggestions in a user-item matrix with significant data sparsity due to the huge number of missing values. To address this sparsity issue, recommendation algorithms should be able to handle missing values efficiently and generate accurate predictions from the remaining data.

Several solutions use trusted connections and user behavior to rectify data sparsity and enhance the overall user preference picture. Incorporating trust, which reflects a user’s belief in the accuracy of another user’s ratings (both explicit and implicit), has been shown to significantly improve the reliability of recommender systems (Guo, 2013). The strength of trust can be calculated based on how closely connected users are in the

social network, typically measured by the number of direct connections or the length of the shortest path between them (O’Donovan and Smyth, 2005).

### 2.2.2 Class Imbalance

Recommender systems are vulnerable to bias amplification during training due to the inherent popularity bias within the data itself. Less popular products have sparser data because popular products are more likely to be reflected in user interactions. This bias may be aggravated by existing methods that depend on similarity between user-item preferences. As the algorithms identify similar users and items based on past interactions, they inherently favor popular products users have already interacted with. This phenomenon creates a twofold issue : it further increases data sparsity for unpopular items and causes an imbalance in the representation of items within the training data. Consequently, recommendations become less accurate, as the system prefers popular products over less popular but potentially relevant ones (Ricci et al., 2010).

The problem of imbalanced data, where training data is not evenly distributed across categories, presents a significant hurdle for recommender systems. For example, in a movie recommendation system, blockbuster movies receive a high number of ratings and interactions, while independent or niche films receive very few. If the model primarily learns from the popular movies, it will tend to recommend these blockbusters to all users, neglecting the less popular films that might be more relevant to certain users. To overcome this obstacle, experts are developing various approaches. These techniques fall into three main categories : data-level methods, algorithm-level methods, and hybrid methods (Johnson and Khoshgoftaar, 2019).

Data-level methods use distinct data sampling techniques to balance the data, ensuring that all categories are adequately represented. Algorithm-level methods, on the other hand, adjust the weights within the model to reduce bias toward the dominant class. Hybrid approaches combine elements from both data-level and algorithm-level methods to achieve optimal results (Johnson and Khoshgoftaar, 2019).

In recent years, several novel techniques have been introduced to tackle class imbalance in recommender systems. For instance, (Cai et al., 2021) proposed a model based on

item-to-item transition patterns within categories, demonstrating that category-specific transitions are more effective than general transitions. Another study, (Chen et al., 2021), focuses on the coexistence of items, creating a system that leverages interactions between items and users. They argue that a co-occurrence neural network provides better feature encoding compared to traditional methods. Finally, (Jiang et al., 2021) suggests a CBF model that dynamically learns knowledge features to capture user preferences. This model integrates knowledge representation based on self-attention to enhance performance.

### 2.2.3 Cold-Start

The “cold-start” problem in recommendation systems refers to the difficulty of making accurate recommendations for new users or items with little to no interaction history. In most practical scenarios, the need to recommend new items without historical user feedback is substantial and poses a significant challenge (Zhu et al., 2021).

The cold start problem manifests in three distinct scenarios : (a) generating recommendations for new users, (b) offering suggestions for new items, and (c) providing recommendations on new items for new users. Each of these scenarios presents unique challenges that necessitate tailored solutions in the field of recommendation systems (Lika et al., 2014).

For example, consider a new user who has just signed up for a streaming service like Netflix. This user has not yet rated any movies or shows, making it difficult for the system to understand their preferences and provide personalized recommendations. Similarly, when Netflix adds a new movie to its library, there is no initial user feedback or ratings to indicate whether the movie is likely to be popular or to whom it might appeal. These are instances of the cold-start problem, where the lack of historical data complicates the recommendation process.

In the context of a standard CF problem, we can better understand the cold-start challenge by examining the user-item interaction data. A collection of  $N$  users  $U = \{u_1, \dots, u_N\}$  and a set of  $M$  items  $V = \{v_1, \dots, v_M\}$  are involved in a standard CF problem. The user input for the items can be represented as a  $N \times M$  preference matrix  $R$ , where  $R_{uv}$  indicates the user  $u$  preference for item  $v$ .  $R_{uv}$  can be understood

as implicit interactions such as plays, views, and purchases, or as explicit interactions defined by the user as rates, likes, dislikes, and so on. In the implicit environment,  $R$  is commonly binary, while in the explicit setting, it usually comprises graded relevance (e.g., 1–5 ratings). If  $R_{uv} = 0$ , then there is no preference information available. The set of users who indicated preference for  $v$  is denoted by  $U(v) = \{u \in U \mid R_{uv} \neq 0\}$  and the set of items that  $u$  expressed preference for is denoted by  $V(u) = \{v \in V \mid R_{uv} \neq 0\}$ . Since no preference data is available at the cold start, when a new user and/or new item is included in the system, we formally define the cold start as the situation in which  $V(u) = \emptyset$  and  $U(v) = \emptyset$  for a given user  $u$  and item  $v$  are both true (Volkovs et al., 2017a).

A number of mitigating techniques have been put out in the literature to help recommendation systems overcome the difficulties caused by the cold start. By using item characteristics to provide recommendations, CBF approaches provide a solution and are especially helpful for new items (Volkovs et al., 2017b). Furthermore, these techniques can take advantage of user attributes or demographic information to better meet the needs of new users (Safoury and Salah, 2013). Hybrid approaches that combine CF with CBF techniques are comprehensive solutions that effectively solve cold start problems for both users and objects (Tahmasebi et al., 2021). Techniques known as discovery-exploitation are crucial because they use tactics such as exploration to learn about new products or users and exploitation to make recommendations based on existing information. Besides, knowledge transfer mechanisms are investigated, which enable the transfer of knowledge based on similarity in content or user behavior from current users or objects to new ones (Volkovs et al., 2017a; Wu et al., 2020; Zhang et al., 2023; Kuznetsov and Kordík, 2023).

#### 2.2.4 Scalability

The fast expansion of e-commerce sites has posed substantial scaling issues for recommendation systems. For these large-scale applications, modern recommendation systems techniques must provide speedy results. While conventional recommendation systems can locate possible neighbors in real-time, the volume of users and items in current e-commerce necessitates searching in a considerably broader pool. This presents a challenge to algorithms, particularly for users with substantial data profiles. Given

that today’s e-commerce sites have hundreds of thousands of products for a single user, it’s easy to see why identifying an appropriate neighbor for that user has become a hard and time-consuming operation.

These issues are particularly visible in filtering algorithms that use nearest-neighbor approaches. The massive rise in users and items necessitates a large increase in computer power needed for these algorithms to function successfully. Scalability is an important consideration for platforms with millions of users and products.

Various techniques can be used to solve scalability problems in recommender systems. A common approach is dimensionality reduction using techniques such as clustering (Abbasi-Moud et al., 2021; Takács et al., 2009). Here users are segmented using a clustering algorithm. Each cluster then serves as a recommendation neighborhood. When a user needs a suggestion, his/her neighborhood is determined within the segment assigned to him/her, and predictions are produced by applying classical filtering algorithms within that segment. This approach provides two main benefits : first it reduces the sparsity of the data set. Furthermore, segmenting the data into smaller pieces drastically cuts down on the amount of time needed to provide predictions. Singular Value Decomposition (SVD) is an additional method for enhancing scalability. SVD is a dimensionality reduction technique that yields a collection of uncorrelated eigenvectors. These eigenvectors encompass both users and items. Users who score similar (but not identical) products will be assigned the same eigenvectors. After decomposing the rating matrix into SVD component matrices, predictions can be made by computing the cosine similarity between user and product representations (Sarwar et al., 2002). By employing these techniques, recommendation systems can better handle the scalability issues posed by the fast-growing number of users and items in e-commerce platforms.

### **2.2.5 Diversity**

Recommendation systems can create difficulty in balancing the desire for both similar and dissimilar recommendations. Ideally, a system should provide accurate recommendations based on user or item similarities and also offer a variety of options that will introduce users to discoveries. However, focusing too much on similar items can lead to a diversity problem where recommendations become repetitive and potentially valuable

niche options are overlooked (Ricci et al., 2010).

For example, consider a user on a music streaming service like Spotify. If the recommendation system only suggests songs that are very similar to the user’s past listening history, the user might only receive recommendations for the same genre or even the same artists. While these recommendations might be highly accurate, they lack diversity and prevent the user from discovering different genres or new artists that they might also enjoy. This raises the concern that prioritizing diversity may come at the expense of accuracy (Zhou et al., 2010).

Diversity can be interpreted on two levels : inter-user diversity, or the variation between recommendation lists, is the ability of an algorithm to return different results to various users. The other one, which we refer to as intra-user diversity (i.e., diversity within a suggestion list), measures how well an algorithm can deliver a variety of items to each user (Zhou et al., 2010). The definition of inter-user diversity ((Zhou et al., 2008)) takes into account the diversity of users’ suggestion lists. The key is to address the diversity issue while maintaining a certain level of accuracy in recommendations. Over-focus on accuracy can lead to over-concentration and the system keeps recommending similar items.

## 2.3 Evaluation Metrics

In this section, we explain commonly utilized performance evaluation metrics in the literature (Chen and Liu, 2017; Lerato et al., 2015; Cremonesi et al., 2010) in the recommendation systems field.

### 2.3.1 Precision@K

Precision@K is the proportion of the sum of the items presented to the user that is suitable for the user to the sum of the items suggested (Chen, Hua, Chang, Wang, Zhang and Kong, 2018).

$$P@K = \frac{\sum \text{Relevant items in K recommendations}}{\sum \text{Items in K recommendations}} \quad (2.4)$$

### 2.3.2 Mean Average Precision@K (MAP@K)

MAP@K evaluates the system's accuracy and ranking accuracy by taking into account the relevance of retrieved items and their position in the ranked list. A higher MAP@K score (from 0 to 1) indicates better performance; 0 means no relevant items were found and 1 indicates perfect retrieval (Chen and Liu, 2017).

$$MAP@K = \frac{1}{M} \sum_{j=1}^M \frac{1}{r} \sum_{k=1}^K Precision@K.rel(k) \quad (2.5)$$

where  $M$  denotes the total user count,  $k$  denotes the rank, and  $rel(k)$  is the relativity function of the  $k$ th rank.

### 2.3.3 Recall@K

Recall@K is represented as the percentage of the sum of items recommended to the user and relevant to the user to the sum of all items that may be suitable to that user (Chen, Hua, Chang, Wang, Zhang and Kong, 2018).

$$R@K = \frac{\sum \text{Relevant items in K recommendations}}{\sum \text{All Possible Relevant Items}} \quad (2.6)$$

### 2.3.4 Normalized Discounted Cumulative Gain (NDCG)

NDCG is rooted in two other metrics; first, Cumulative Gain (CG) and second, Discounted Cumulative Gain (DCG) (Järvelin and Kekäläinen, 2002). If the relevant value is calculated for each recommended item, the CG would be the sum of the calculated relevance scores. However, CG does not bear in mind the order of the offered items. To solve the problem, the DCG metric has emerged. In this metric, highly relevant items

are penalized if they are suggested to the user in a lower order. Finally, NDCG is the regularized form of the DCG at each location for a selected p-value (Wang et al., 2013).

$$CG_k = \sum_{i=1}^k rel_i \quad (2.7)$$

where  $k$  is the rank and  $rel$  is the relevance score.

$$DCG_k = \sum_{i=1}^k \frac{2^{rel_i} - 1}{\log_2(i + 1)} \quad (2.8)$$

$$NDCG_k = \frac{DCG_k}{IDCG_k} \quad (2.9)$$

$$IDCG_k = \sum_{i=1}^{|Rel_k|} \frac{2^{rel_i} - 1}{\log_2(i + 1)} \quad (2.10)$$

where  $rel$  is the list of items sorted by relevance.

### 2.3.5 Hit Rate

The primary goal of the Hit Rate is whether the algorithm recommends at least one relevant item to a particular user. When there is at least one relevant item in the top recommendation list for user  $u$ , it is indicated by a Hit Rate of 1. A Hit Rate of 0, on the other hand, indicates that none of the recommended items are relevant to the user  $u$  (Ferrari Dacrema et al., 2021). Hit rate does not indicate the number of relevant items that were recommended or their ranking inside the top-k list because it is a binary indicator. The hit rate can be expressed using the formula below :

$$HitRate@k(u) = \mathbb{I}[|rel(u) \cap reck(u)| > 0] \quad (2.11)$$

When the condition enclosed in brackets is true,  $\mathbb{I}$ , an indicator function, returns 1, and when it is false, it returns 0. Relevant items for user  $u$  are denoted by  $rel(u)$  and the set of recommended items for the user is  $u$  denoted by  $reck(u)$ .

### 2.3.6 Mean Reciprocal Rank (MRR)

MRR calculates the effectiveness of a recommendation system based on the user’s ranking of relevant items. It takes into account where the first relevant suggestion is in the list. Increased MRR signifies that recommendations usually include more pertinent items at the top, which makes it simpler for users to discover them. MRR takes the minimum reciprocal rank among all relevant items in the top-k recommendations (Tamm et al., 2021). This allows the metric to penalize scenarios where related items are spread across the list. However, MRR does not take into account the relevance of items other than the first relevant one. A list with two highly relevant items listed lower down might have the same MRR as a list with a moderately relevant item listed first.

$$MRR@k(u) = \frac{1}{\min_{i \in rel(u) \cap reck(u)} rank(i, u)} \quad (2.12)$$

where relevant items for user  $u$  are denoted by  $rel(u)$  and the set of recommended items for user  $u$  denoted by  $reck(u)$ .

## 2.4 User-based Personalized Recommendation Frameworks

In this subsection, we will examine the algorithms in the scope of this study. First of all, we have incorporated the Bayesian Personalized Learning algorithm, notwithstanding its deviation from the graph-based methodology. This departure arises from the utilization of the Bayesian Personalized Ranking (BPR) (Rendle et al., 2009) algorithm within the loss functions of certain graph-based methods, such as Light Graph Convolutional Neural Network (LightGCN) (He et al., 2020), Neural Graph Collaborative Filtering (NGCF) (Wang et al., 2019), and Self-Supervised Graph Learning for Recommendation (SGL) (Wu et al., 2021). We posit that the inclusion of the BPR algorithm is crucial for enhancing the comprehension of graph-based algorithms. In this study, we have specifically used NGCF, LightGCN, and SGL for the experiments.

### 2.4.1 Bayesian Personalized Ranking (BPR)

In traditional machine-learning-based recommendations, the training set is built using user-item pairs (Hu et al., 2008). The primary purpose of BPR is to improve the order of items for each user based on implicit feedback, such as user-item interactions (e.g., clicks, purchases, and views), rather than explicit numerical ratings. The method is intended to capture user preferences for item ranking rather than anticipate exact scores. If the user has already picked an element, the class label will be positive; otherwise, the class label will be negative.

Latent factor vectors, denoted as  $p_u$  and  $q_i$ , are associated with each user  $u$  and item  $i$ , respectively. According to Eq 2.13, the model predicts that user  $u$  will select item  $i$  as the dot product of their latent factor vectors (Ricci et al., 2010).

$$\hat{r}_{ui} = p_u^T \cdot q_i \quad (2.13)$$

The primary objective of the BPR optimization procedure is to maximize the likelihood of observed paired preferences. As stated in (Rendle et al., 2009), this is achieved by accounting for a triplet (user, positive item, negative item), in which the user prefers the positive item to the negative item.

The BPR loss for a triplet  $(u, i, j) \in D$ , where  $i$  is a positively interacted item and  $j$  is a randomly sampled non-interacted item, is defined as follows in 2.14 :

$$\text{BPR Loss}(\Theta) = - \sum_{(u,i,j) \in D} \ln \sigma(\hat{r}_{uij}) + \lambda_{\Theta} \cdot \|\Theta\|^2 \quad (2.14)$$

$D$  is the set of observed interactions,  $\hat{r}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$  is the difference in predicted scores for items  $i$  and  $j$  for user  $u$ ,  $\sigma(x)$  is the sigmoid function which is calculated as  $\sigma(x) = \frac{1}{1+e^{-x}}$ . Lastly,  $\|\Theta\|^2$  is the squared L2 norm of the model parameters.  $\lambda_{\Theta}$  is a regularization parameter to prevent overfitting.

The BPR loss is assigned to give greater points to positively interacting items ( $\hat{r}_{ui}$ ) than to randomly selected non-interacting items ( $\hat{r}_{uj}$ ). The difference between these scores,

$\hat{r}_{ui} - \hat{r}_{uj}$ , is converted into a probability using the sigmoid function. The logarithm of this probability is then taken, and the optimization procedure minimizes the negative log-likelihood, which encourages the model to maximize the score difference between interacting and non-interacting items.

BPR aims to learn model parameters that maximize the likelihood of such observed preferences whose equation is given in Eq. 2.14, via stochastic gradient descent. This likelihood function represents a user’s discretion for an item over another. Here,  $\Theta$  is the model parameter vector to be inferred, which captures the particular relationship  $>_u$  of a user  $u$  and his item preferences. The reformulation of the training data not only allows for ranking-focused optimization, it also tries to solve the inequality between positive and negative samples in the training data.

#### 2.4.2 Neural Graph Collaborative Filtering (NGCF)

The premise that users with similar behaviors are likely to have similar preferences underpins many recommendation techniques. Leveraging consumers’ historical actions and their opinions regarding the products they have engaged with enhances the understanding of user-item interactions (Wang et al., 2015; He et al., 2017).

In this context, two main components are crucial : (1) user-item interaction representation, which relies on the produced embeddings, and (2) the embeddings themselves, which convert users and items into vector representations. (Wang et al., 2019) argues that current approaches (He et al., 2017; Koren et al., 2009; Wang et al., 2015) are unable to provide embeddings that capture enough information. This inadequacy stems from the fact that the embedding functions employed in these models primarily consider the attributes of either users or items, but do not adequately account for the connections involving users and items. To manage this limitation, (Wang et al., 2019) proposes the utilization of *high-order connectivity* obtained from interactions between users and items into the embedding function via the neural network model. This approach naturally encodes collaborative signals within the structure of the interaction graph, thereby rectifying the aforementioned issue which is illustrated in Fig. 2.3. For instance, consider the user for whom recommendations are to be made is node  $u_1$ . The

term high-order connectivity refers to the path that connects node  $u_1$  to any other node that is more than one hop away in terms of path length. This high-order connectivity is significant as it encapsulates valuable semantic information that conveys collaborative signals. An illustration of this concept can be seen in the path  $u_1 \leftarrow i_2 \leftarrow u_2$  which implies a behavioral resemblance between  $u_1$  and  $u_2$  due to their interaction with  $i_2$  (Wang et al., 2019).

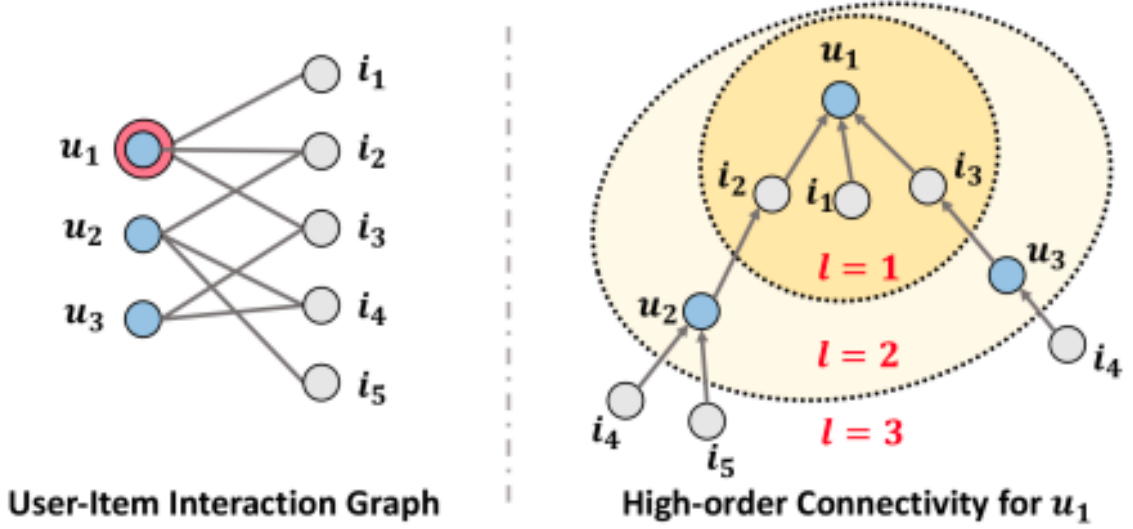


FIGURE 2.3 – A representation of the high-order connectivity and user-item interaction graph. Image courtesy from (Wang et al., 2019)

Two important processes have been devised to guarantee the proper capture of collaborative signals and enhance embedded data of users and items : (1) message construction and (2) message aggregation.

Information intended for understanding dissemination (i.e. message embedding) is denoted as  $m_{u \leftarrow i}$ , which is formally represented in Equation 2.15.

$$m_{u \leftarrow i}^{(l)} = \frac{1}{\sqrt{|N_u| |N_i|}} \left( W_1^{(l)} h_i^{(l-1)} + W_2^{(l)} \left( h_i^{(l-1)} \odot h_u^{(l-1)} \right) \right) \quad (2.15)$$

where  $W_1, W_2 \in \mathbb{R}^{d' \times d}$  are the trainable weight matrices.  $d'$  is the transformation size. In this equation, user and item embeddings are considered together with element-wise product  $\odot$ .

In order to refine the embeddings, they are aggregated as follows :

$$h_u^{(l)} = \text{LeakyReLU} \left( m_{u \leftarrow u}^{(l)} + \sum_{i \in N_u} m_{u \leftarrow i}^{(l)} \right) \quad (2.16)$$

The self-connection of node  $u$ , denoted as  $m_{u \leftarrow u}^{(l)} = W_1^{(l)} h_u^{(l-1)}$ , is an important consideration as well as the messages produced from its neighboring nodes  $N_u$ . This self-connection mechanism enables the retention of the original feature information.  $h_i^{(l-1)}$  denotes the item representation that arises from the preceding iterations of message-passing, capturing the information conveyed by its  $(l-1)$ -hop neighbors. This item representation subsequently supplies the portrayal of user  $u$  at layer  $l$ .

Upon completing the  $L$  propagation layer, we are left with multiple representations for user  $u$ , which are represented as  $\{h_u^{(1)} \dots h_u^{(L)}\}$ . These representations highlight the signals that are sent across different connections and have different roles in communicating user choice. Thus, diverse representations ought to be focused in order to create the optimal embedding for a user, as shown in Equation 2.17. Similarly, the same process is applied to obtain multiple representations for each item.

$$h_u^* = h_u^{(0)} \oplus \dots \oplus h_u^{(L)}, \quad h_i^* = h_i^{(0)} \oplus \dots \oplus h_i^{(L)} \quad (2.17)$$

where  $\oplus$  is the concentration function, which can be modified by any other aggregation functions such as sum-pooling, max-pooling, etc.

The inner product is finally used to estimate the user's disposition regarding the target item.

$$\widehat{y_{ui}} = (h_u^*)^T h_i^* \quad (2.18)$$

where  $\widehat{y_{ui}}$  stands for the score for item  $i$  is recommended to user  $u$ .

NGCF optimizes the pairwise BPR loss (Rendle et al., 2009) in order to estimate model parameters. This loss function considers the relative order of observed and unobserved interactions between users and items as in Equation 2.14.

### 2.4.3 Light Graph Convolutional Neural Network (LightGCN)

LightGCN draws inspiration from the NGCF method discussed in the preceding section. NGCF originated from the GCN (Kipf and Welling, 2016) framework and incorporates the nonlinear activation function  $\sigma(\cdot)$ , as well as feature transformation matrices  $W_1$  and  $W_2$  (as defined in Eq. 2.15). However, the effectiveness of the nonlinear activation function and the previously mentioned transformation matrices is questioned in CF, where user and item IDs are the main information available and node attributes are limited. The authors' research suggests that adding these factors can potentially complicate the learning process. It has been demonstrated that when all of the nodes' properties are known, the nonlinear activation function and matrices utilized by NGCF for feature transformation yield better results.

In LightGCN, on the other hand, feature transformation and nonlinear activation are not used; instead, it uses the simple weighted sum adder. The (Wang et al., 2019) propagation rule (also known as the graph convolution process) in LightGCN is defined as follows :

$$h_u^{(l+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u||N_i|}} h_i^{(l)}, \quad h_i^{(l+1)} = \sum_{u \in N_i} \frac{1}{\sqrt{|N_u||N_i|}} h_u^{(l)} \quad (2.19)$$

The enhanced embeds for the user  $u$  and item  $i$  are spread over the following layer  $l$  and are indicated by  $h_u^{(l+1)}$  and  $h_i^{(l+1)}$ , respectively.  $N_u$  stands for the group of items with which user  $u$  interacts. Users who associate with item  $i$  are identified by the term  $N_i$  and the  $\frac{1}{\sqrt{|N_u||N_i|}}$  is the normalization term.

The aggregation procedure in LightGCN only includes the target node's linked neighbors and excludes the target node itself, which is a critical distinction to make. Focusing just on the relationships between nodes in the network and excluding the target node entirely speeds up the CF process and simplifies the model.

In the LightGCN framework, the only trainable parameters are the embeddings in the first layer, denoted as  $h_u^{(0)}$  for each user and  $h_i^{(0)}$  for each item. After determining the

embeddings, Eq 2.19 is used to compute them in the following layers. Furthermore, the authors propose integrating the embeddings produced from each layer using the following procedure to generate the final depiction of a specific user or item :

$$h_u = \sum_{l=0}^L \alpha^{(l)} h_u^{(l)}; \quad h_i = \sum_{l=0}^L \alpha^{(l)} h_i^{(l)} \quad (2.20)$$

where  $\alpha^{(l)}$  reflects the layer's relevance and can be interpreted as a hyperparameter that requires tuning.

The authors provide three key points to justify the combination of layers in GCNs. Firstly, as the total layer number increases in the GCN, using only the embedding information in the last layer without using the information in the previous layers will cause information loss (Li et al., 2018). Secondly, distinct node representation characteristics may be revealed by various embeddings at various levels. Thus, the combination of layers can contribute to a more precise depiction of the node (Wang et al., 2019). Lastly, they stress that the best way to illustrate the impact of graph convolution with self-connections is to combine embeddings from several layers using a weighted sum. This method is essential for improving the performance of GCNs.

The prediction produced by the suggested method is expressed as the inner product of the final user and item representations. This inner product serves as the ranking score for the recommendation generation process.

$$\widehat{y}_{ui} = h_u^T \cdot h_i \quad (2.21)$$

where  $\widehat{y}_{ui}$  stands for the score for item  $i$  is recommended to user  $u$ .

For model optimization, LightGCN uses BPR loss 2.14, as in NGCF method.

#### 2.4.4 Neighborhood-based Neural Collaborative Filtering (NNCF)

Matrix factorization (MF) is widely used in recommender systems to uncover latent properties in user-item interactions by representing the original matrix as a product

of lower-dimensional matrices (Koren et al., 2009). However, traditional MF’s linear factorization may fail to capture the complexity of user-item interactions, affecting performance in certain scenarios.

To address this, researchers have integrated deep learning with recommendation systems, such as neural collaborative filtering (NCF) and autoencoder-based recommenders (Wang et al., 2015). These methods enhance the modeling of complex interactions but, like MF, struggle to identify strong relationships among closely related items in sparse data (Koren, 2008).

NNCF aims to maintain the flexibility of neural network models while improving the exploitation of localized information to better understand interaction data, particularly in scenarios requiring strong relationships among a small set of closely related items (Bai et al., 2017).

We can define the user-item interaction as a bipartite network formally defined as  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where the vertex set  $\mathcal{V}$  consists of users and items nodes, i.e.,  $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$ , and the edge set  $\mathcal{E}$  consists of the two-way interaction relations between users and items, and  $(u, i) \in \mathcal{E}$  only if the interaction label  $y_{u,i} = 1$ . NNCF uses this network to not only learn overall patterns but also zoom into local connections, named the “interaction-based neighborhoods.”

The significance of these neighborhoods lies in their integral role within the framework of NNCF. When NNCF provides recommendations to users, it transcends general trends by delving into their local environment. Specifically, NNCF examines the preferences of the user’s “close friends” (similar users) or preferences in relation to similar items. This localized information serves to improve recommendations by making them more personalized and relevant to each user.

There are various methods for building these neighborhoods. A simple approach involves aggregating all directly connected neighbors of a user or item within the network; this represents entities that the user directly interacts with, similar friends who share similar movie tastes. But NNCF extends its capabilities beyond this basic approach. NNCF employs advanced methods like community exploration to locate cohesive clusters of closely connected nodes in the network. Specifically, the (Traag, 2015) network’s modularity is optimized through the application of the Louvain technique. Within

bigger organizations, these defined communities function as more inclusive neighborhoods that share common interests or preferences.

To generate the latent vector  $\mathbf{v}_{u,i}$  for user-item pair  $(u, i)$ , the authors encode user-item interactions by applying a linear kernel to the latent vectors. Convolution and max pooling operations affect the latent vectors  $\mathbf{v}_u^{(N)}$  and  $\mathbf{v}_i^{(N)}$  for user and item neighborhoods, respectively. Equation 2.22 illustrates how the authors propose to combine latent vectors into a single vector representation to incorporate neighborhood information into user-item interactions.

$$\tilde{\mathbf{v}}_{u,i} = \mathbf{v}_{u,i} \oplus \mathbf{v}_u^{(N)} \oplus \mathbf{v}_i^{(N)} \quad (2.22)$$

where  $\mathbf{v}_{u,i}$  is the latent vector for the user-item pair  $(u, i)$ ,  $\mathbf{v}_u^{(N)}$  is the latent vector for the neighborhood of user  $u$ , and  $\mathbf{v}_i^{(N)}$  is the latent vector for the neighborhood of item  $i$ .

Multi-Layer Perceptron (MLP) processes the generated vector  $\tilde{\mathbf{v}}_{u,i}$  using a hidden layer with non-linear activation functions. The MLP output layer provides the anticipated interaction between the user and the item. The authors choose to use Rectifier Linear Unit (ReLU) as the activation function.

For implicit tasks involving 0 or 1 user element interaction, the NNCF framework can also be applied. Equation 2.23 defines the loss of the set of training samples  $\{(u, i, y)\}$ , where  $y$  is the actual interaction label. For model optimization, NNCF uses the cross-entropy loss as follows :

$$\mathcal{L} = - \sum_{(u,i,y) \in \text{training set}} [y_{u,i} \log(\hat{p}_{u,i}) + (1 - y_{u,i}) \log(1 - \hat{p}_{u,i})] \quad (2.23)$$

where  $\hat{p}_{u,i}$  is the predicted interaction probability from the MLP, and  $N$  is the number of training samples.

### 2.4.5 Self-Supervised Graph Learning for Recommendation (SGL)

The main goal of the collaborative recommendation is to extract high-quality representations of users and items from interaction data. From utilizing a single ID or interaction history, representation learning on user-item graphs has progressed to employing higher-order neighbors for the suggestion. This led to the successful application of GCNs in recommendation systems, such as LightGCN. Though effective, current GCN-based recommender models face difficulties, such as handling sparse data, which leads to imprecise predictions for less well-liked “long-tail” items. Furthermore, their reliance on labeled data restricts their applicability in situations where explicit feedback is expensive or rare, and they show vulnerability to misleading signals from noisy interactions, for example accidentally clicking the user. Self-Supervised Graph Learning (SGL) solves these problems by making use of unlabeled data, automatically detecting negatives that are difficult to interpret, and building noise resilience, which opens the door to suggestions that are more accurate and understandable (Wu et al., 2021).

SGL addresses these issues by utilizing unlabeled data, a.k.a implicit feedback automatically detecting hard-to-interpret negatives and building resilience to noise, thereby enhancing the accuracy and interpretability of recommendations (Wu et al., 2021).

However, effective model learning is hampered by the high degree of sparsity that implicit interactions frequently display. To address these issues, SGL introduces data augmentation techniques applied to the graph structure (Wu et al., 2021).

The first technique is called “Edge Dropout” which involves randomly removing a percentage of edges in order to simulate missing interactions and force the model to give priority to strong connections. The second one is “Node Dropout” which involves temporarily removing nodes, which might be users or items, to avoid overfitting to local associations and promote the discovery of wider patterns. The last technique for data augmentation, Random Walk, involves rearranging edges while maintaining node degrees, which enhances resilience to particular interaction orderings (Wu et al., 2021).

After generating the augmented graphs, the model gains the ability to provide consistent node representations in each of these various perspectives. One phase in the learning process is to identify positive pairs and negative pairs via contrastive learning. The

main goal of contrastive learning is to teach the model to distinguish between similar and dissimilar examples. It does this by pushing the representations of dissimilar pairs (negative pairs) further apart and pulling the representations of similar pairs (positive pairs) closer together. This process helps create a more distinct and meaningful embedding space, where similar nodes are positioned near each other and dissimilar nodes are positioned farther apart (Wu et al., 2021).

We can express it mathematically as follows :

Consider the node  $v_i$  is denoted by the embedding in the original graph as  $f(v_i)$  and the same node is embedded in an augmented version of the graph as  $f'(v_i)$ . To minimize the distance between embeddings of positive pairs  $(v_i, v_i)$  and maximize the distance between embeddings of negative pairs  $(v_i, v_j)$  the contrastive loss is defined as follows :

$$L_{\text{user}}(v_i, v_j) = \sum_{u \in U} -\log \frac{\exp(\text{sim}(f(v_i), f'(v_i))/\tau)}{\sum_{v \in U} \exp(\text{sim}(f(v_i), f'(v_j))/\tau)} \quad (2.24)$$

Here, the hyper-parameter,  $\tau$ , is the temperature in softmax, and  $\text{sim}(a, b)$  is a similarity function (e.g., dot product or cosine similarity) between embeddings  $a$  and  $b$ . The authors also compute the item-side contrastive loss. The combined version of the  $L_{\text{user}}$  and  $L_{\text{item}}$  is the total contrastive loss, which is calculated as follows by 6 :

$$L = L_{\text{user}} + L_{\text{item}} \quad (2.25)$$

### 3 OUR SOLUTION FOR DATA SPARSITY

One of the main challenges in machine learning tasks is data sparsity, which occurs when there are not enough available data points to capture underlying relationships effectively. When it comes to recommender systems, many possible items and few user-item interactions are the main causes of data sparsity. This leads to problems such as misleading recommendations and poor model performance.

In this chapter, a novel paradigm for representing data sparsity in recommender systems—called Multi-Granular Projection and Comparison (M-GraP) is proposed. We will explore the specifics of this framework, such as the selected level of detail in the data and the way the user-item interactions are represented on the network. The chapter will then go on to detail our experimental setup as well as the performance measurement strategy that we applied.

#### 3.1 Data Granularity and Network Representation

The level of detail (number of attributes) available in the data representation, in particular, determines how effective recommender systems are. Richer data allows for finer-grained information, while data with fewer qualities allows for coarser representations (Grčar et al., 2006). Consider a user searching for a “bag” on Amazon as in the Figure.

3.1. While the search itself suggests a basic level of detail (“bag”), the recommendation system needs to delve deeper. Should it simply recommend “bags” or consider additional features like material (leather, canvas), size (tote, clutch, backpack), or brand (recognizing user preference for a specific brand)? This question of *data granularity*, or the level of detail in item representation, significantly impacts how well recommender systems capture user preferences and recommend relevant items.

Based on Amazon’s recommendation algorithm, the search results shown in the Figure 3.1 currently include bags with different colors, fabric types, and brands. We can clearly see that Amazon’s algorithm generates recommendations in several degrees of

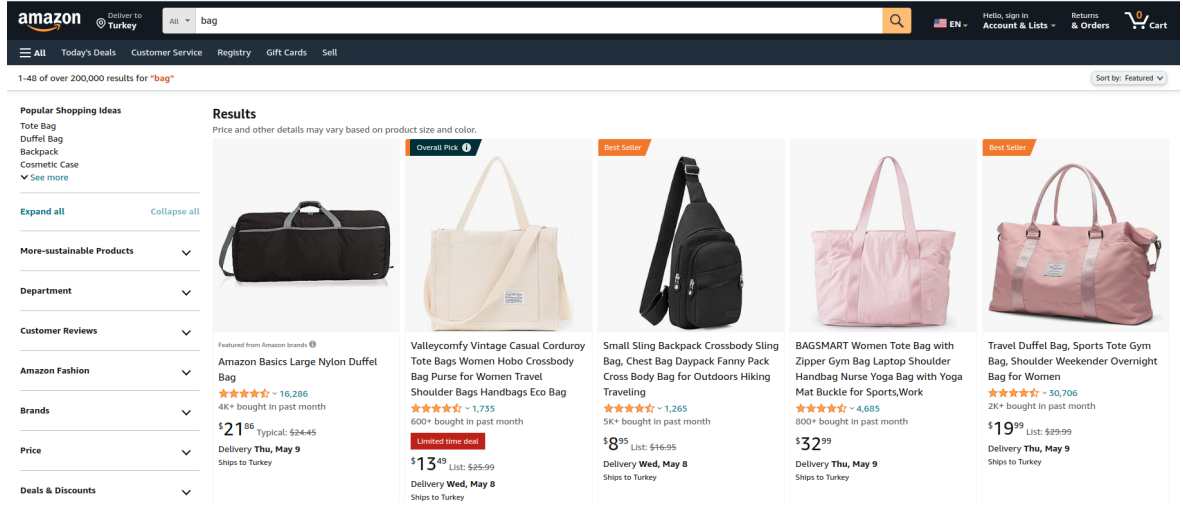


FIGURE 3.1 – Results for “bag” search on Amazon.com

detail. This highlights how important it is to have rich data that records different item characteristics. By learning the user’s preferences for these aspects, the recommender system can make more personalized and relevant recommendations.

From the perspective of recommendation algorithms, using the existing features of the items or users to be recommended for recommendation can be achieved with CBF methods, but their success is limited. They disregard user-item interaction behavior, which is essential for the precise suggestion, and instead concentrate only on the similarity between the user and the item based on attributes (Fayyaz et al., 2020). Additionally, CF methods are generally more successful than CBF methods because they take user-item interaction behavior into account (Da’u and Salim, 2020). However, they often struggle with data sparsity, which is situations where user-item interactions are limited.

In this part of the thesis, in order to alleviate the data sparsity issue in CF methods, our work delves into the heart of a profound philosophical question : How can we capture the essence of an entity through its representation ? What constitutes the necessary and sufficient properties for an accurate description ? Is there an inherent trade-off between comprehensiveness and specificity ? This framework emerges from grappling with these fundamental issues from an experimental point of view.

For example, let us think about explaining a bag to a forthcoming buyer. We need to decide, via an epistemological lens, what degree of specificity is necessary to provide a veridical depiction (one that is accurate and true to reality). Does the term “bag” alone

suffice, or does adding details like color (“red bag”), size, design (“red tote bag”), material (“red leather tote bag”), or brand (“red leather tote Chanel brand bag”) improve the accuracy of the depiction?

To address the preceding questions, we characterize the user-item interaction as a bipartite graph. A bipartite graph is a graph  $G = (V, E)$  where the set of nodes  $V$  can be divided into two disjoint sets  $V = V_1 \cup V_2$ , such that each edge  $(u, v)$  in  $E$  connects a node  $u$  in  $V_1$  to a vertex  $v$  in  $V_2$ , and there are no edges between nodes within the same set  $V_1$  or  $V_2$ . In other words,  $E \subseteq V_1 \times V_2$  is the set of edges where each edge  $(u, v)$  connects a node  $u \in V_1$  to a vertex  $v \in V_2$ .

Previously, the granularity concept was introduced by the Information Granularity in (Pedrycz, 2018). In this thesis, we leverage this concept by constructing network representations of the same items using different attribute sets. This allows us to capture user-item interactions at various levels of detail. We propose a multi-level bipartite graph representation to capture fine-grained relationships between users and items based on different attribute combinations. We denote each level of the representation, namely each graph as  $G_{L_k} = (V_1, V_{L_k}, E_{L_k})$ , that corresponds to interactions based on a specific set of  $K$  attributes (where  $K$  represents the number of attributes used). Each graph shares a common set of user nodes, denoted by  $V_1$ . However, the item nodes, represented by  $V_L$  in each level, differ based on the attributes used for that level. A lower level focuses on a smaller set of core attributes, while a higher level incorporates a broader range of attributes. We aim to understand user preferences better and increase recommendation accuracy with the proposed new experimental design by analyzing user-item interactions at these different levels.

To illustrate, consider a user who purchased a ‘red bag’ on an e-commerce platform like Amazon. At Level 1, we have users are connected to corresponding “bag” nodes. At Level 2, separate nodes are created for each color variation of the bag. Edges are established only between the user and the “red bag”. As we move to Level 3 (e.g., “channel brand red bag”), additional nodes are created for each combination of attributes (brand, channel, color), and the user is linked to the specific “channel brand red bag” she purchased.

This example shows how edges and item nodes change with each level. Formally, as the number of attributes ( $k$ ) increases, the number of item nodes in set  $V_{L_i}$  (representing item nodes at Level  $i$ ) will also grow compared to the previous level ( $V_{L_{i-1}}$ ).

While adding new nodes at each level might appear to increase sparsity, the key lies in attribute effectiveness. Suppose an attribute is relevant to interaction behavior (e.g. price). In that case, the network density remains relatively constant as the level increases, since products with that key attribute are more likely to be purchased. This allows us to identify important item attributes through sparsity analysis potentially.

Our research aims to identify the most informative representations for user-item interactions via bipartite graphs. We focus on constructing user-item interaction data as efficient bipartite graphs where each item is represented necessarily and sufficiently based on its attributes. This aims to provide CF algorithms with the most impactful data representations, ultimately leading to improved recommendation performance.

We investigated how varying the detail of user-item interactions affects the performance of recommender systems. In our experiment, we used Last.fm music recommendation data to explore how the level of detail in user listening history influences recommendation accuracy. The detailed explanation of node creation and linking is provided in Section 3.2. This framework can be extended beyond music recommendations. By leveraging the concept of granularity, we can incorporate various user and item attributes into different recommender systems, such as user-product recommendations in e-commerce or user-movie recommendations. This allows CF algorithms to exploit these richer representations, potentially mitigating data sparsity issues.

### 3.2 Experimental Design

In this section, we describe our experimental design to solve the granularity selection dilemma. In our design we have 5 main steps : (1) Data Augmentation, (2) Level Projection, (3) Recommendation, (4) Degrading Recommendations, and (5) Performance Measurement. Our proposed experimental design is illustrated in Figure 3.5

### 3.2.1 Data Augmentation

In order to achieve the best learning results, modern machine learning algorithms require large datasets (Mumuni and Mumuni, 2022). However, in the field of recommender systems, the nature of interaction data makes it difficult to obtain sufficient information on every user. That is why, (Wu et al., 2021), for instance, proposes to apply the data augmentation techniques, proven to enhance performance and widely employed in computer vision and natural language processing, which could be extended to the realm of user-item interaction data. Specifically, traditional data augmentation methods used in computer vision and natural language processing tasks do not apply to our setting due to the discrete and interrelated structure of user-item interactions.

Considering user-item interaction data as a bipartite graph as we already explained in the previous section, multiplexing user or item nodes is the main technique to provide data augmentation. This method is a crucial way to supplement data while also increasing the level of detail in the augmented data.

In terms of improving data representation, we hypothesized that there is/are user-item representation(s) that will enable the description of user and/or item, that is, ideal data granularity and our main goal is to find the most sufficient and necessary attributes for data representation.

Figure. 3.2 provides an example of how our suggested data augmentation technique is put into practice. We focused on music artists as items and analyzed user interactions at different levels of granularity. In the figure 3.2, users are represented by nodes labeled  $u_n$  (where  $n$  is the total number of users), and artists are represented by nodes labeled  $i_m$  (where  $m$  is the total number of artists).

The figure's most important feature is the way that links between artists and users are made at various degrees of granularity (degrees 1, 2, and 3). There is no data modification involved in Level 1 as each user is connected directly to the artist whom they have listened.

The degree of specificity (granularity) in user-artist interactions rises as we approach Level 2. This allows us to capture richer insights beyond just listening history. At this

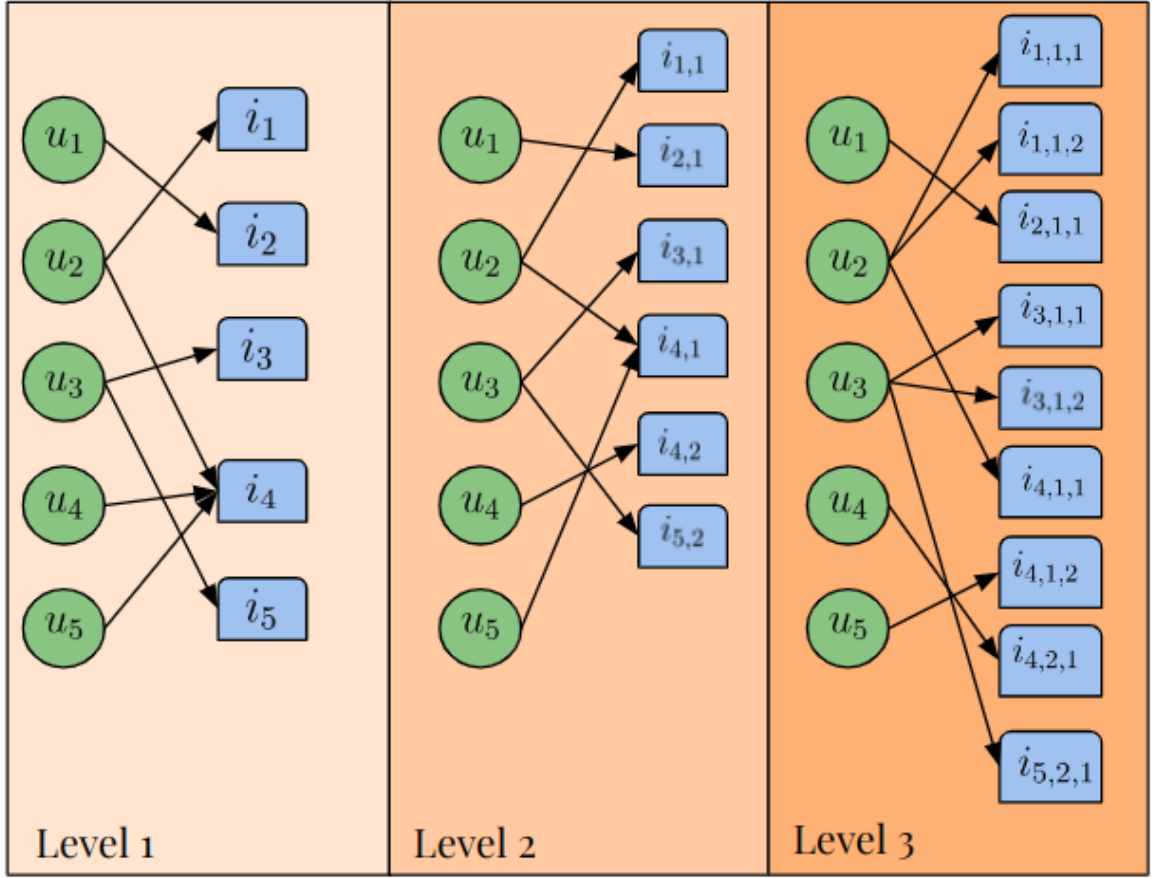


FIGURE 3.2 – Data Augmentation : We progressively enrich the representations of items by incorporating additional attributes into their node definitions at each level.

level, we incorporate a weight for each user-artist pair, representing the listening count. For illustration, let us look at users  $u_2$ ,  $u_4$ , and  $u_5$  who are listeners of  $i_4$  artist in Level 1. That being said, at Level 2, users might have different listening counts. Users  $u_2$  and  $u_5$  may have a lower listening count (weight) for artist  $i_4$  than user  $u_4$ , indicating that they may listen to  $i_4$  less frequently, and thus they connected with different nodes. In this way, the recommender system may be able to generate more individualized recommendations with this extra listening frequency data.

Level 3 introduces an additional data granularity beyond just listening history and frequency. At this level, we incorporate artist tags into the user-artist interaction. These tags can reflect genres, moods, or other characteristics associated with the music given by the users. Take Level 2, for instance, where users  $u_2$ ,  $u_4$ , and  $u_5$  are all listening to artist  $i_4$  with a given listening count. At Level 3, we may find that  $u_2$  listens to  $i_4$ , which is tagged as “alternative” and “rock”, and  $u_5$  listens to  $i_4$ , which is tagged as “pop”. More

artist tag information can help the recommendation system understand user preferences for certain genres or styles of music, resulting in more focused recommendations.

In addition, we incorporated the temporal dimension into the user-artist interaction at Level 4. We take each listening event’s month and year into account in addition to the data in Level 3. This enables the recommender system to perhaps identify changing user preferences over time or seasonal trends. For instance, imagine at Level 3, we discover user  $u_2$  listens to artist  $i_1$  with tags like “rock” and “alternative”. At Level 4, we might see that  $u_2$  listens to  $i_1$  more frequently during the summer months. This additional information about listening time (month and year) can help the recommender system understand temporal patterns in user listening habits, potentially leading to recommendations that adapt to seasonal preferences. The sample data for each level is available in Table 3.1, Table 3.2, Table 3.3, and Table 3.4, respectively.

TABLE 3.1 – Sample data used for Level 1

| userID | artistID | Timestamp |
|--------|----------|-----------|
| 1      | 5        | 08-22     |
| 2      | 12       | 06-21     |
| 3      | 85       | 09-23     |

TABLE 3.2 – Sample data used for Level 2

| userID | artistID | Timestamp |
|--------|----------|-----------|
| 1      | 5_1      | 08-22     |
| 2      | 12_2     | 06-21     |
| 3      | 85_1     | 09-23     |

TABLE 3.3 – Sample data used for Level 3

| userID | artistID | Timestamp |
|--------|----------|-----------|
| 1      | 5_1_1_1  | 08-22     |
| 2      | 12_2_5_2 | 06-21     |
| 3      | 85_1_2_3 | 09-23     |

TABLE 3.4 – Sample data used for Level 4

| userID | artistID | Timestamp |
|--------|----------|-----------|
| 1      | 5_1_1_1  | 08-22     |
| 2      | 12_2_5_2 | 06-21     |
| 3      | 85_1_2_3 | 09-23     |

### 3.2.2 Level Projection

Due to the significant differences in information content across various granularity levels (Level 1-4), evaluating recommender system performance at each level requires independent procedures. To ensure fair comparisons, we need to address the differences in information content across levels. Instead of directly comparing results, a common ground needs to be established. This can be achieved by a data transformation process, which we call “*level projection*”, that enriches the lower-granularity data with information from the higher-granularity levels. In simpler terms, instead of using data points with less detail (lower level), we can replace them with more detailed data points that still encompass the original information

Illustratively, consider Level 1, where user  $u_2$  is recorded to have listened to the artists  $i_1$  and  $i_4$ . Nonetheless, using the most detailed data available, it can be shown that  $i_1$ ’one listening count was subject to two separate tagIDs, 1, and 2, “Alternative”, and “Rock” respectively. Furthermore, there are two separate weights and tags connected to  $i_4$ . As a result,  $i_4$  can be identified as four separate nodes, while  $i_1$  can be effectively represented as two separate nodes. The level projection process creates connections between  $u_2$  and all the nodes representing artists he has listened to (Figure. 3.3). This allows the model to access more detailed information about those artists.

Projecting from Level 2 to Level 3 involves similar techniques, which result in datasets that consistently represent the same node information and can therefore be compared.

### 3.2.3 Recommendation

We employed three recommender system algorithms (NGCF (Wang et al., 2019), LightGCN (He et al., 2020), SGL (Wu et al., 2021)) to generate top-10 artist

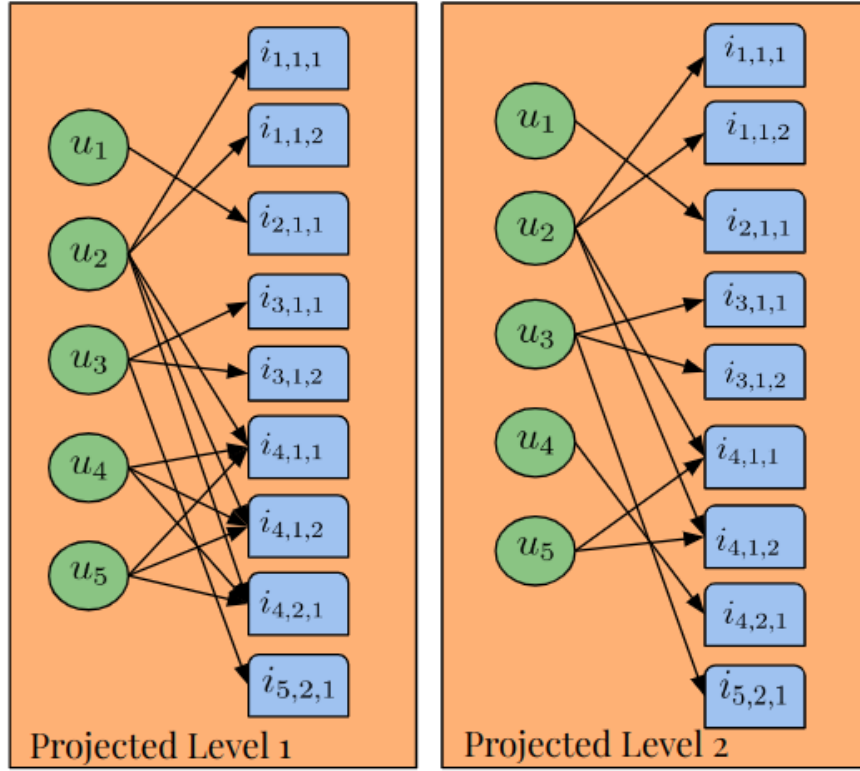


FIGURE 3.3 – Level Projection

recommendations on the resulting user-item interaction bipartite graph. This bipartite graph is obtained after projecting each network representation to its most granular level. Notably, the number of these bipartite graphs corresponds to the number of attributes (or levels) used.

### 3.2.4 Degrading Recommendations and Performance Measurement

Our investigation necessitates evaluating the effectiveness of the proposed approach in mitigating data sparsity. It is not possible to directly compare the recommendations made by models trained on data with different predicted levels. There are two reasons for this : (1) Models trained on data with different levels of granularity and projected to the most granular level have fundamentally different conceptions of "items". For example, a music recommendation system trained on artists, with data projected to the finest granularity, may recommend completely different artists than a system trained on listening counts within artists, with data projected to the finest granularity. Direct comparisons are meaningless as they operate on fundamentally different data units. (2)

Models with more detailed data in their unprojected form may have access to richer information than models with less detailed data. This inherent information asymmetry makes it difficult to isolate the effect of granularity on performance.

Directly comparing recommendations made on data with different granularities is challenging. Two alternative approaches can be considered :

1. **Degraded Recommendation Evaluation** : We first train recommender system models using the data projected to the most granular level (e.g., including attributes such as weights, tagIDs, and temporal information), which ensures all models have access to the richest information available. After training, the recommendations generated by the models undergo a “degradation” process where details are systematically removed until they align with the least detailed level—namely, just the artists. This degradation process strips away the additional information present in higher granularities, creating a standardized set of recommendations suitable for comparison.

To evaluate the effectiveness of these standardized recommendations, we use common recommender system metrics such as Precision@K or Recall@K. Since all recommendations are standardized to the same level of detail, this approach allows for a fair comparison of performance across different data granularities.

Figure 3.4 illustrates the recommendation degradation process for our scenario involving artist recommendations. In the left table, we have recommendations from a model using projected data across Levels 1 to 3 (L1-L3), containing detailed information including artist, weight, and tag information for each item. To facilitate proper evaluation, we degrade these recommendations to the most coarse level, which focuses solely on the artist IDs. The right table shows the degraded recommendations, where each entry simplifies to just the artist ID ( $i_n$ ), removing finer details like weight and tagID for a standardized comparison.

2. **Level-Specific Recommendation with Aggregation (Attempted but Discarded)** : In this strategy, we train separate models for each data level, operating without the projection between levels. Each model independently generates a list of the top-10 recommendations for each user at its respective level of granularity.

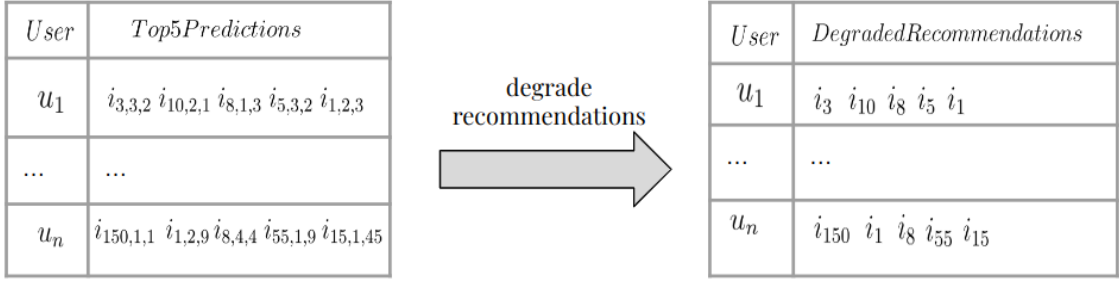


FIGURE 3.4 – Degrading Recommendations

To facilitate a fair comparison across the different granularities, we aggregate these recommendations to the most detailed level. This means that the recommendations from a coarser level are expanded to encompass all variations that exist at the finest level.

For example, consider a scenario where a model operating at the lowest granularity level recommends the artist  $i_X$  for a user. Suppose at the highest level of granularity, this artist  $i_X$  is represented by 10 distinct variations (such as different weights or tags). In our approach, recommending artist  $i_X$  at a lower level is equivalent to recommending all 10 variations of artist  $i_X$  at the highest level. Therefore, for the purpose of comparison, we consider all these variations as part of the recommendation list.

When we apply this method of aggregation, the initial top-10 list of recommendations can quickly expand and become dominated by various combinations of the same artists' different variations. This could lead to an overrepresentation of certain artists in the final aggregated recommendation list, thereby skewing the evaluation and potentially making it less meaningful. Due to this limitation and the potential distortion in evaluation accuracy, we did not include the Level-Specific Recommendation with Aggregation approach in this thesis.

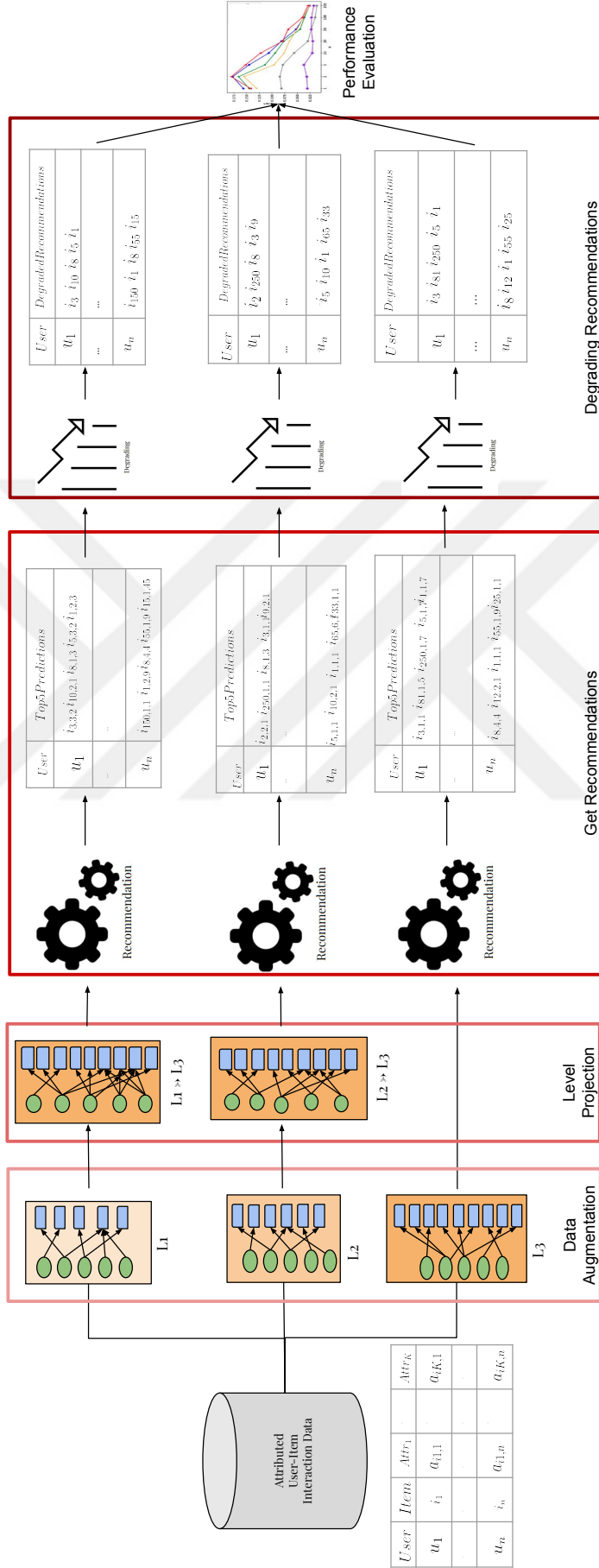


FIGURE 3.5 – Data Granularity Integration Flow Chart

## 4 OUR SOLUTION TO CLASS IMBALANCE

Class imbalance occurs when one class (usually the minority class) has significantly fewer data points than the majority class. This is a recurring problem for machine learning algorithms. This imbalance can lead to the underperformance of models that benefit the dominant class over the minority class (Johnson and Khoshgoftaar, 2019).

The use of negative sampling strategies to address class imbalance in the recommendation systems field is covered in detail in this chapter. We start by looking into current negative sampling methods. We examine the development of negative sampling techniques, from their inception to the state of the art. Following that, we provide a range of negative sampling strategies created in the literature. Consequently, in this thesis, we present our novel network science-oriented negative sampling strategy designed to address class imbalance by taking into account the user-item interaction topology.

### 4.1 Negative Sampling

The concept of negative sampling initially emerged as a simplified alternative to noise contrastive estimation (NCE) (Mikolov, Sutskever, Chen, Corrado and Dean, 2013). NCE is a technique that transforms the problem of probability estimation into a binary classification task. By doing so, it efficiently approximates the partition function of an unnormalized distribution, which is essential for normalizing probabilities in probabilistic models (Mikolov, Chen, Corrado and Dean, 2013).

Word2vec, a popular model for word embeddings, leverages negative sampling to enhance its training process. In Word2vec, the goal is to predict words that surround a given word (context) to capture word relationships. Typically, this involves calculating the likelihood of each word in the vocabulary being the next word, often using a softmax function. However, calculating the softmax function for large vocabularies is computationally expensive because it requires evaluating the probability for every word.

In order to address this slowness and computational overload, negative sampling strategies emerged. The computing load is considerably decreased by concentrating on a smaller group of target words. The methodology simply takes into account positive context words and negative words, not the complete vocabulary. Here, negative words refer to a small number of words drawn at random from the lexicon, and positive context words are the words around the target word. Negative sampling significantly enhances training efficiency by classifying these word pairs as either 'likely to appear together' (positive examples) or 'unlikely to appear together' (negative examples) (Mikolov, Sutskever, Chen, Corrado and Dean, 2013).

Negative sampling highlights its value by offering several benefits in machine learning algorithms :

- It significantly reduces training time and computing costs by employing fewer typical negative examples.
- This approach helps to solve the class imbalance issue in real-world data by preventing models from being biased towards the dominant class.
- Models can learn finer distinctions between data points through negative sampling, especially when “hard” negatives that resemble positive examples are included, resulting in better learning (Yang et al., 2020).

As remarked, negative sampling simplifies training a machine learning model by differentiating between positive examples (relevant pairs) and a small set of carefully chosen negative examples (irrelevant pairs) drawn from a larger pool of candidates. A loss function that aims to maximize similarity between positive pairs and minimize similarity between negative pairs is used to formalize this strategy :

$$L = l(x, x^+, x^-), \quad x^- \sim p_n(x^-) \quad (4.1)$$

where  $l(\cdot)$  is a specific loss function in various domains. In different application domains, the specific loss function may vary based on various tasks and datasets. We can encapsulate them as follows (Yang et al., 2024) :

- **Pair-based sampling** : BPR loss ensures that the positive sample in a pair is ranked higher (has a higher log-likelihood) than the negative sample (Rendle et al., 2009).
- **Triple-based sampling** : Hinge loss focuses on triples (anchor, positive, negative) where the anchor’s similarity to the positive example is at least one margin greater than its similarity to the negative example (Rennie and Srebro, 2005).
- **Single positive, multiple negatives** : Cross-entropy loss helps determine a single positive sample from a set of negative samples (Rennie and Srebro, 2005).
- **Comparative learning** : InfoNCE loss is specifically designed for comparative learning frameworks that aim to maximize the contrast between positive pairs and negative pairs (Oord et al., 2018).

Recommendation systems specifically utilize negative sampling techniques to address the data sparsity and class imbalance issues and improve model training efficiency (Ying et al., 2018). In such systems, during the training process, for each observed positive interaction  $(u, i)$ , a negative item  $j$  is sampled randomly or using a strategy that ensures it is unlikely to be a positive interaction.  $j$  is then used in the triplet  $(u, i, j)$  to calculate the pairwise ranking loss in Eq. 4.2.

$$\text{Loss}_{uij} = -\log(\sigma(\hat{r}_{uij})) \quad (4.2)$$

$\hat{r}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$  is the difference in predicted scores for items  $i$  and  $j$  for user  $u$ .  $\sigma(x)$  is the sigmoid function. To encourage the model to accurately rank positive interactions higher than negative ones, the goal is to update the model parameters such that the projected score for the positive interaction  $\hat{r}_{ui}$  is greater than the anticipated score for the negative interaction  $\hat{r}_{uj}$ .

#### 4.1.1 Negative Sampling Evolution

Negative sampling approaches, inspired by NCE, have been developed to improve model performance by addressing issues related to imbalanced and sparse data (Wu, Gao, Zang, Han, Wang and Hu, 2022).

The first and simplest approach to negative sampling is random negative sampling (RNS). Here, the model randomly selects a subset of data points from the entire data to act as negative examples for a given positive node. This method contributes to a significant reduction in the computational cost of training when compared to accounting for all possible negative data values. Numerous tasks, including recommender systems (Wang et al., 2019; He et al., 2020) and graph embedding (Xiao et al., 2016), have shown success with RNS. In these applications, RNS helps overcome difficulties related to a lack of negative data like class imbalance and data sparsity (Pan et al., 2008).

As datasets grow larger and models become more complex, researchers have realized there are limitations with RNS. It is possible that randomly selected nodes do not adequately represent “true” negativity, in which case the training data may be ineffective. The introduction of popularity-based negative sampling (PNS) was aimed at addressing this issue. This strategy chooses negative instances based on how frequently they occur in the data. The assumption is that frequent nodes are less likely to connect to the positive node we consider. PNS focuses on these more likely “true negatives”, making negative sampling more relevant to the particular data set. A common approach samples negatives with a probability proportional to  $3/4$  of their frequency; this has demonstrated its effectiveness in areas such as natural language processing (Mikolov, Sutskever, Chen, Corrado and Dean, 2013) and graph embedding learning (Perozzi et al., 2014; Grover and Leskovec, 2016).

RNS and PNS, however, share a further drawback in that they employ static negative examples during training, which may reduce performance. Hard negative sampling (Hard NS) techniques have been presented as a solution by selecting negative samples that the model finds challenging at its current stage of training (Zhang et al., 2013). These “hard negatives” provide the model with a larger learning signal, which increases the model’s capacity to distinguish between positive and negative examples. Hard NS has been used for a variety of applications, including word embedding (Chen, Yuan, Jose and Zhang, 2018), recommendation systems (Ying et al., 2018; Huang et al., 2021), and object detection (Shrivastava et al., 2016).

Generative Adversarial Networks (GANs) are a recent development in negative sampling techniques. In GAN-based negative sampling (GAN-based NS), two models are placed against one another in an adversarial training environment (Wang et al., 2017;

Park and Chang, 2019). The generator, one of the models, seeks to generate realistic negative examples that are hard for the trained model’s discriminator to discern from genuine positive ones. This method is very beneficial when the dataset contains very few actual negative examples.

Another technique is in-batch negative sampling (In-batch NS), which is a successful method in unsupervised visual representation learning. In-batch NS uses other samples within the same batch as negative samples for the positive sample evaluated (Chen et al., 2020). This is computationally efficient since it does not require additional steps for negative sample production or selection. However, the quality of negatives may be reduced due to batch size. Smaller groups may not exhibit sufficiently various difficulties, limiting the model’s ability to learn complicated representations. To solve this, in literature, some studies introduce hard negatives into the learning framework (Kalantidis et al., 2020; Robinson et al., 2020).

#### 4.1.2 Negative Sampling Algorithms

For the training process of the machine learning model, two fundamental decisions need to be addressed in the context of negative sampling : (1) how to generate a candidate set for negative examples and (2) how to select an appropriate negative sampling model. There are five techniques for selecting candidate negatives (Yang et al., 2024) :

- **Global Selection** : This is the simplest method that takes the entire dataset into account. Although training data has a wide range of data points, it can result in poor learning because it contains negative and irrelevant examples. For instance, word2vec (Mikolov, Sutskever, Chen, Corrado and Dean, 2013) uses the entire dictionary as a pool of possible negative samples, and LightGCN (He et al., 2020) also uses all unseen elements as a pool.
- **Local Selection** : This method selects negative instances from a specific subset rather than a pool of potential negative examples. Because the selection is performed from a defined subset, the computing workload is decreased. For example, ANCE (Xiong et al., 2021) selects the top-k negative examples based on the query as the pool of negative samples, or DNS (Zhang et al., 2013) samples a random subset as the pool of negatives for computational probabilities.

- **Mini Batch Selection** : This method uses pre-loaded data for training. Additional samples from the current batch are used as negatives while analyzing the positive sample. This is computationally efficient, but the batch size may limit the quality of the negatives.
- **Hop Selection** : This technique works only with graph-structured data. It utilizes  $k$ -hop neighbors, nodes within  $k^{th}$  distance, as potential negative instances. With large datasets, it could not be feasible because it uses the graph structure.
- **Memory-Based Selection** : This method maintains a memory bank that stores negative examples from past training iterations or batches. This allows for a large and diverse pool of negativity, which is particularly useful for comparative learning tasks.

The following subsections provide a comprehensive explanation of negative sampling techniques.

#### 4.1.2.1 Static Negative Sampling (Static NS)

Static NS is typically employed as a fundamental baseline since it offers a quick and easy solution. Static NS provides several advantages over other heuristic sampling techniques, including speed, stability, and model independence. The disadvantage of such techniques is that low-quality negatives are sampled because the negative sampling distribution is fixed during the whole training phase and the sampled negatives are not adaptive for each positive pair. In this section, we will touch on PNS and RNS.

**Random Negative Sampling (RNS)** , also known as “Uniform Negative Sampling” is the most common negative sampling technique and is frequently utilized as the default sampling method to assess the suggested model’s performance in optimization with negative sampling (Yang et al., 2024). The word “uniform” suggests that all potential negative samples that the user has not interacted with are uniformly and randomly selected. During training, it chooses negative examples at random to fix the imbalance between interacted and non-interacted elements. This technique suffers the problem of inefficiency since many negative samples might not be the actual negatives, which does not provide any meaningful information (Sun et al., 2019).

Mathematically speaking, for user  $u$ , consider that we have a set of items  $I_u$  that the user has not interacted with. From this set,  $n$  items are uniformly sampled. We can represent it as follows :

$$I_{u,\text{neg}} \sim \text{Uniform}(I_u) \quad (4.3)$$

where  $I_{u,\text{neg}}$  is the set of sampled negative items.

**Popularity-based Negative Sampling (PNS)** samples negative items based on item popularity distribution. We can formalize the selection as follows :

$$p(v|u) \propto \text{pop}(v)^a, \quad (4.4)$$

explains the probability  $p(v|u)$  of selecting item  $v$  given user  $u$ . In this equation,  $\text{pop}(v)$  represents the popularity of item  $v$ , which could be measured by metrics such as the number of interactions or the number of times the item has been viewed. The parameter  $a$  is a tuning parameter that alters the influence of item popularity on the selection probability. This parameter  $a$  can significantly affect the model’s performance by controlling how much weight is given to popular items when sampling negative examples (Yang et al., 2023). Commonly,  $a$  is set to  $3/4$  in various domains such as network embedding (Mikolov, Chen, Corrado and Dean, 2013; Perozzi et al., 2014; Grover and Leskovec, 2016; Tang et al., 2015), or graph representation learning (Hamilton et al., 2017).

#### 4.1.2.2 Hard Negative Sampling (Hard NS)

Hard NS improves model training by concentrating on negative examples that are challenging to find. It is based on the data values that appear positive but are actually negative, which is called “hard negatives”. Unlike prior approaches that used random negative samples, hard NS emphasizes cases the model is uncertain about or searches the feature space for data points near the decision boundary. We can categorize the Hard NS into two main strategies : (1) dynamic negative sampling (DNS), which looks for hard negatives within the raw training data, and (2) mixture-based DNS, which generates new negative examples straight from the model’s embedding space (Yang et al., 2024).

**Dynamic Negative Sampling (DNS)** is a technique that selects informative negative samples during training based on the current state of the model. There are three main approaches to DNS :

1. **Anchor-based DNS** focuses on finding informative negative examples similar to a selected “anchor” point. This strategy creates negatives based on how far they are (in the feature space) from the anchor. Different measures such as inner product scores (e.g. in tasks like word embedding (Chen, Yuan, Jose and Zhang, 2018)), distance measures in feature space (Wu et al., 2017), or PageRank scores (eg. in tasks that use graph-structured data (Ying et al., 2018)) can be used to define “far” depending on the scenario.
2. **Positive-based DNS** addresses a limitation of anchor-based DNS. Anchor-based techniques are effective, but they undervalue the impact of positive examples. This method focuses on negatives in the feature space that are more closely aligned with the positive samples. Positive-based DNS enhances model performance by concentrating on these tricky negatives. For instance, in max sampling (Rao et al., 2016), the negatives that have the highest similarity to the positive instance are selected. On the other hand, two-stage sampling (Tran et al., 2019) first narrows down candidate negatives and then chooses the most similar one among those candidates to the positive sample.
3. **Anchor-Positive DNS** combines the advantages of both anchor-based and positive-based DNS. The objective is to select negatives based on a certain loss function, such as triple loss or Weighted Approximate-Rank Pairwise loss (WARP) loss, that are both “confusing” to the model in comparison to the positive sample and close to the anchor (Yang et al., 2024).

**Mixture-based DNS** Inspired by Mixup (Zhang et al., 2018), mixture-based DNS generates synthetic hard negatives directly in the embedding space, without depending on pre-existing data points. It initially selects a collection of negative samples from the data. Then it employs a function to strategically mix the embeddings of various negatives, resulting in a new synthetic negative sample. MixGCF (Huang et al., 2021), for instance, uses information propagation in graphs to synthesize negatives from multiple locations and also mixes positive information to strengthen the negatives.

#### 4.1.2.3 GAN-based Negative Sampling

This method uses Generative Adversarial Networks (GANs) to create informative negative examples that challenge the model during training. It succeeds in using two networks : (1) generator network, and (2) discriminator network. The generator network invents new negative examples to “fool” a separate discriminator network. Conversely, the discriminator network attempts to discriminate between genuine positive examples and the generator’s created negative ones. By continuously testing the discriminator, the generator gets better at producing negative instances that are more intricate, and as a result, the model can get more informative negative instances which speeds up learning and enhances performance overall. However, training GANs can be difficult, unstable, and frequently requires pre-training of the networks, which contradicts the original goal of faster training with negative sampling. The development of a reliable training method for GAN-based negative sampling is an ongoing research topic. Furthermore, adding a network (the generator) increases the computational load and training time (Yang et al., 2024).

#### 4.1.2.4 Auxiliary-based Negative Sampling

Auxiliary-based negative sampling uses extra information to improve the quality of negative samples. This extra information can be related to data, how the data is connected (e.g. graph), or even how often positive and negative instances are used together. While this approach can improve the negative sample quality and model performance, it needs more data, and that can be computationally expensive. (Yang et al., 2024).

#### 4.1.2.5 In-batch Negative Sampling

In-batch NS leverages the mini-batch structure during training. It employs additional samples within the same mini-batch as implicit negatives, rather than explicitly sampling negatives for every positive case. This method works well with larger batches. However, its effectiveness depends on batch size. There are three main types of In-batch NS (Yang et al., 2024) :

**Basic In-batch NS** treats all non-positive pairs in the mini-batch as negatives. For instance, SGL (Wu et al., 2021) integrated self-supervised graph learning into recommendation and created two types of in-batch neural networks for the auxiliary task. Despite its simplicity, this method may produce false negatives in the negative samples it creates.

**Debiased In-batch NS** tackles the false negative issue by adjusting the weights of negative samples during training. It improves performance over basic in-batch NS (Chuang et al., 2020).

**Hard In-batch NS** focuses on finding and using “hard negatives” within the batch, which are more informative for training (Xiong et al., 2021).

## 4.2 Network Science Oriented Hard Negative Sampling

In this section, we describe our proposed network science-inspired negative sampling techniques. In contrast to traditional approaches that frequently handle data points independently, we propose to examine the data’s intrinsic network structure by treating it as a bipartite graph and went to reveal hidden links. Our goal is to find more difficult negative examples by including the network structure into the negative sampling approach. This allows us to exploit network properties, particularly path lengths and densities, to identify informative negative samples for training.

Our network science-oriented approaches prioritize identifying “hard” negative samples – those that closely resemble positive examples. While easy negatives (highly dissimilar) are generally learned well, focusing on hard negatives improves the model’s sensitivity to subtle variations, potentially leading to a more robust and accurate decision-making process.

By leveraging network features for Hard NS, we aim to achieve two key benefits. First, the model can be trained more effectively by focusing on the most challenging negative examples, leading to faster convergence and better performance. Second, incorporating network features can improve the model’s generalization by exploiting the structural

relationships in the data, enabling it to perform well on unseen data with similar network properties.

After this introduction, we shall discuss our suggested strategies in more detail. We'll begin by discussing the Path-Length Negative Sampling (PL-NS), which includes two strategies : Long-Distance Weighted and Quartile-Based Negative Sampling, respectively. Next, we will explore the Path-Density Negative Sampling (PD-NS), followed by a detailed explanation of the Diffusion Distance-based Negative Sampling (DD-NS).

#### 4.2.1 Path-Length Negative Sampling (PL-NS)

The PL-NS strategy leverages the concept of “the shortest distance” within the network structure between users and items to identify informative negative examples for training the recommendation system. Since nodes with very short paths might be considered less informative negatives as they share some similarity with the positive pair, this method focuses on generating negative examples that are connected to the positive node by longer paths. We produce increasingly difficult negative samples by giving priority to nodes that are farther apart in the network topology. We hope to reinforce the model’s decision-making bounds and enhance its capacity to distinguish between comparable data points by adding these negative examples.

For each proposed strategy, we leverage user-item interaction data to construct a bipartite graph. Each user and item is represented by a unique identifier (node) in the graph, and an edge connects a user to an item they interacted with. This creates a network representation of user-item relationships. Our goal is to choose items from the bipartite graph that have longer paths (distances) from a user. There are two primary steps involved :

##### 1. Data Preparation

- (a) **Shortest Path Calculation :** The `all_pairs_shortest_path_length` function is utilized from the `NetworkX` library to compute the shortest path length between every pair of nodes (users and items) in the graph. This function determines how few edges, or interactions, are needed to link any two nodes. The output is a dictionary with a key-value pair for each user-item

and the matching shortest path length. The pseudo-code of this built-in function is explained in Algorithm 1 and Algorithm 2.

## 2. Candidate Selection Strategy

(a) **Creating Distance-Weighted Lists** : For each user, we create a list of candidate negative items for sampling. This list construction can be done in two different ways :

- i. **Long-Distance Weighted Selection** : Using the distance information from the distance dictionary, we multiplied each item by its distance from that user in the candidate list. For instance,  $i_1$  will appear five times in the candidate list if the distance between  $u_1$  and  $i_1$  is equal to five. This assigns a higher weight to items with longer paths, making them more likely to be included in the list. As a result, items having longer paths are given a higher weight and are therefore more likely to be included in the list. We exclude any positive items from the list following weighing in order to stop them from being selected as negatives. Finally, we randomly chose one element from the created list to serve as the negative sample for each positive item the user interacted with.
- ii. **Quartile-Based Selection** : In this strategy, we start by removing the user's positive items from the initial dictionary of shortest path lengths. Next, we sort the remaining distances in ascending order to create a sorted list. We then focus on Quartile 1 (Q1), representing the median of the first half of the sorted distances, and Quartile 2 (Q2), representing the overall median. These specific points in the sorted list correspond to particular path lengths away from the user. Finally, we randomly select items at these chosen distances and assign them as negative items.

The illustration of the proposed approach can be found in Figure 4.1

In summary, this technique gives precedence to selecting negative items in the network that are located farther away from the user via longer paths. While quartile-based selection directly targets certain distance ranges through list sorting, the long-distance weighted selection uses weighting to favor distant items. Through the use of network

---

**Algorithm 1:** Single Source Shortest Path Length
 

---

**Data:** Graph  $G$ , source node  $source$

**Result:** Shortest path lengths from  $source$  to all other nodes

**Function** `single_source_shortest_path_length( $G$ ,  $source$ )` :

```

   $lengths \leftarrow \{\}$ ;
   $nextlevel \leftarrow [source]$ ;
   $seen \leftarrow \{source\}$ ;
   $level \leftarrow 0$ ;
   $lengths[source] \leftarrow 0$ ;
  while  $nextlevel$  is not empty do
     $level \leftarrow level + 1$ ;
     $thislevel \leftarrow nextlevel$ ;
     $nextlevel \leftarrow []$ ;
    foreach node  $v$  in  $thislevel$  do
      foreach neighbor  $w$  of  $v$  in  $G$  do
        if  $w$  not in  $seen$  then
          add  $w$  to  $seen$ ;
          add  $w$  to  $nextlevel$ ;
           $lengths[w] \leftarrow level$ ;
        end
      end
    end
  end
  return  $lengths$ ;

```

---

---

**Algorithm 2:** All Pairs Shortest Path Length
 

---

**Data:** Graph  $G$

**Result:** Shortest path lengths between all pairs of nodes in  $G$

**Function** `all_pairs_shortest_path_length( $G$ )` :

```

  shortest_path_lengths  $\leftarrow \{\}$ ;
  foreach node  $n$  in  $G$  do
    | lengths_from_n  $\leftarrow$  single_source_shortest_path_length( $G, n$ );
    | shortest_path_lengths[ $n$ ]  $\leftarrow$  lengths_from_n;
  end
  return shortest_path_lengths;

```

---

structure (path lengths) in the negative sampling process, both strategies seek to enhance the model’s capacity to learn fine differences between positive and negative data points.

#### 4.2.2 Path-Density Negative Sampling (PD-NS)

This approach incorporates path density between users and items. Path density refers to the relative number of paths connecting a user to an item within the user-item interaction network. Here, we investigate the relationship between path density and the difficulty of capturing an item as a “hard negative” example for the recommender system. This method involves two main steps :

##### 1. Data Preparation

- (a) **Path Density Calculation** We would like to examine the number of paths between user and item nodes for this approach. While computing the exact number of paths between each user and item (path density) is computationally expensive due to its #P-completeness (Roberts and Kroese, 2007), we propose to use random walks to estimate this value. This estimated path count is then used to inform the selection of negative samples during the candidate generation process. Our strategy simulates random walks on the user-item interaction network. It starts at a user (source) and walks for a set length on the network. If the walk reaches a specific item (target) during the walk, it’s considered a successful path. Many random walks are

conducted by the function, and the number of successful walks is recorded. This estimated count provides an approximation of the number of paths between the user and the item. We did our experiments for `num_walks=100` and `max_walk_length = 10`. The pseudo-code of the path density calculation function is in the Algorithm 3.

---

**Algorithm 3:** Estimate Paths
 

---

**Data:** Graph  $G$ , source node  $source$ , target node  $target$ , number of walks

$num\_walks$ , maximum walk length  $max\_walk\_length$

**Result:** Estimated number of paths from  $source$  to  $target$  in  $G$

**Function**

```

estimate_paths( $G, source, target, num\_walks, max\_walk\_length$ ) :
   $num\_paths \leftarrow 0$ ;
  for  $walk \leftarrow 1$  to  $num\_walks$  do
     $current\_node \leftarrow source$ ; for  $step \leftarrow 1$  to  $max\_walk\_length$  do
      if  $current\_node = target$  then
        |  $num\_paths \leftarrow num\_paths + 1$ ; break;
      end
       $neighbors \leftarrow$  neighbors of  $current\_node$  in  $G$ ; if  $neighbors$  is
        empty then
          | break;
        end
         $current\_node \leftarrow$  randomly select a neighbor from  $neighbors$ ;
      end
    end
  end
  return  $num\_paths$ ;

```

---

## 2. Candidate Selection Strategy

- (a) **Max Path Density Selection** Items having a high path density, i.e. high number of paths, from a particular user are the focus of this technique. It is hypothesized that items with a large number of paths will be harder for the model to identify as “hard negatives”. As a result, we include the items in the candidate list by multiplying them in proportion to their path density. In this way, we increase the probability of items with high path density being

selected as negative items.

- (b) **Min Path Density Selection** Items with a low path density, i.e. low number of paths, on the other hand, are thought to be likely to constitute “hard negatives”. We use a method to balance their inclusion probability with the high-path items such that these items remain represented in the candidate list. In the candidate list, we increase the items in inverse proportion to the path density. In this way, we increase the chances of items with low path density being selected as negative items.

The illustration of the proposed approach can be found in Figure 4.2

The final candidate list for each user is constructed by removing the positive items (items the user interacted with) and selecting a random negative item from the remaining list. This strategy departs from conventional negative sampling techniques by taking into account the network structure of user-item interactions and the possible influence of path density on collecting “hard negatives”. Our goal is to increase the efficiency of negative sampling for recommendation systems by integrating path density information.

#### 4.2.3 Diffusion Distance-based Negative Sampling (DD-NS)

Our method measures node similarity by utilizing the idea of information propagation inside a network. To investigate the network topology and comprehend information flow across it, this method uses random walks. A random walk is a stochastic process in which a walker moves randomly across the network, from one node to another. We can assess how easy or difficult it is for information to spread throughout the network using random walks.

Mathematically, an undirected graph  $G = (V, E)$  has  $V$  set of nodes and  $E$  set of edges. Each node  $v \in V$  represents a position in the network, and each edge  $(u, v) \in E$  represents a connection between nodes  $u$  and  $v$ .

Beginning at an initial node  $(v_0)$ , the random walk iteratively advances, with equal probability at each step, to its neighbors. Let  $N(v)$  be the set of neighbors of the node

$v$ . The transition probabilities can be defined as follows (Masuda et al., 2017) :

$$P(v_i = v_j | v_{i-1} = v) = \begin{cases} \frac{1}{|N(v)|}, & \text{if } v_j \in N(v) \\ 0, & \text{otherwise} \end{cases} \quad (4.5)$$

Until a stopping requirement is satisfied or for a certain number of steps, the random walk proceeds. Analyzing random walks can provide a comprehensive picture of network activity, including the relative importance of different network regions, their organizational structure, and their vulnerability to disruptions (Bertagnolli and De Domenico, 2021).

In this thesis, we use a special type of random walk distance, known as the *diffusion distance*, to measure the similarity between nodes in the network. Since this distance measure takes into account both the physical connections between nodes and the ease of information flow, we expect that more informative negative examples will be identified using this similarity measure. This makes it possible to grasp information diffusion patterns and network interactions at a more comprehensive level (Bertagnolli and De Domenico, 2021).

We used the `get_distance_matrix` function from the `diffudist` package in R to determine the negative instances. This function creates a matrix with each item denoting the node pairs' diffusion distance. The steps involved in the procedure are as follows (Bertagnolli and De Domenico, 2021) :

1. **Adjacency Matrix Calculation** : Let us consider, a graph  $G$  with set of nodes  $V = \{v_1, v_2, \dots, v_N\}$ . The adjacency matrix,  $\mathbf{A}$  is an  $N \times N$  symmetric matrix such that  $A_{ij} = A_{ji} = 1$  if and only if  $\{i, j\} \in E$  (there is an edge between nodes  $i$  and  $j$ ), and  $A_{ij} = A_{ji} = 0$  otherwise.
2. **Node Degree Calculation** : A node's degree, represented as  $k_i$ , is the total number of neighbors of node  $v_i$ . It is computed as  $k_i = \sum_{j=1}^N A_{ij}$ . The degrees of all nodes are collected into the diagonal matrix  $\mathbf{D}$ . The `get_distance_matrix` function assumes that any node without edges does not exist in the graph. If such nodes exist, the convention  $(k_i)^{-1} = 0$  is used.
3. **Transition Matrix Calculation** : It is calculated as  $\mathbf{T} = \mathbf{D}^{-1}\mathbf{A}$ . Specifically,

$T_{ij} = \frac{A_{ij}}{k_i}$  is the probability of moving from node  $i$  to node  $j$ . If there is an edge connecting  $i$  and  $j$ ,  $T_{ij}$  can not be equal to zero.

4. **Continuous-Time Markov Chain :** (Masuda et al., 2017) defines a continuous-time variant of the random walk, which is inspired by the discrete-time random walk. The forward equation, which explains how the walker’s probability distribution varies with time, governs the random walk’s continuous-time version :

$$\frac{d}{d\tau}\mathbf{p}(\tau) = -\mathbf{p}(\tau)\tilde{\mathbf{L}} \quad (4.6)$$

with  $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$  being the (random walk) normalized Laplacian of  $G$ .

5. **Solution of Master Equation :** Given the initial condition  $\mathbf{p}(\tau = 0) = \mathbf{e}_i$ , the solution of the master equation (Eq. 4.6) at time  $\tau > 0$  is  $p_{ij} = \left(e^{-\tau\tilde{\mathbf{L}}}\right)_{ij}$ , representing the probability of finding the random walker at node  $j$  at time  $\tau$  given it started at node  $i$ .
6. **Diffusion Distance :** It measures the distance between two posterior probability distributions of finding the random walker at nodes  $i$  and  $j$  at time  $\tau$ , and calculated as below (Bertagnolli and De Domenico, 2021) :

$$D_\tau(i, j) = \|\mathbf{p}(\tau|i) - \mathbf{p}(\tau|j)\|_2 \quad (4.7)$$

Given that the walker started at node  $i$ ,  $\mathbf{p}(\tau|i)$  is the probability (being at  $j$ ) and describes how likely it is that it will be at node  $j$  after time  $\tau$ .  $\mathbf{p}(\tau|i)$  is calculated by taking the  $i$ -th row of  $p_{ij}$ . This matrix captures the network’s connections and how easily the walker can move between nodes.  $\tau$  is the diffusion time parameter that controls the “spread” of the diffusion process in the network.

In simpler terms, the function calculates how different the starting locations affect the walker’s likelihood of being at various places after some time ( $\tau$ ). A larger diffusion distance indicates the starting locations more significantly influence the walker’s future position. By changing the starting points (users or items) in our diffusion distance calculations, we analyzed how information flows through the user-item network. In

addition to that, we analyzed the time parameter  $\tau$  in our experiments, which shows how information propagates over time.

In this study, for each user, we calculate one negative item for every positive item using diffusion distance. We followed 4 different strategies to find the most informative negative items. The illustration of the proposed approach can be found in Figure 4.3

1. **Farthest Node Selection** We select the most distant node from the positive node as a negative item.
2. **Nearest Node Selection** We select the nearest node from the positive node as a negative item.
3. **Long-Distance Weighted Selection** Based on the calculated diffusion distance, we first normalized the distances in specific ranges. Then, using the normalized distances, we weighted items as explained in the sub-section 4.2.1 approach of the Long-Distance Weighted Selection strategy. Accordingly, items farther away (having a higher distance value) are weighted more heavily. This means they appear more frequently in a list of candidate negative examples. For each positive item, one item is randomly selected from the filtered candidate list (excluding the positive item itself) to be the negative sample.
4. **Quartile-Based Selection** After eliminating the positive items' diffusion distances from the distance list, we sort the remaining distances in ascending order. Instead of weighting, specific positions in the sorted list are chosen. Then, we choose negative examples randomly from the items located at Q1 and Q2 distances from the user in the sorted list.

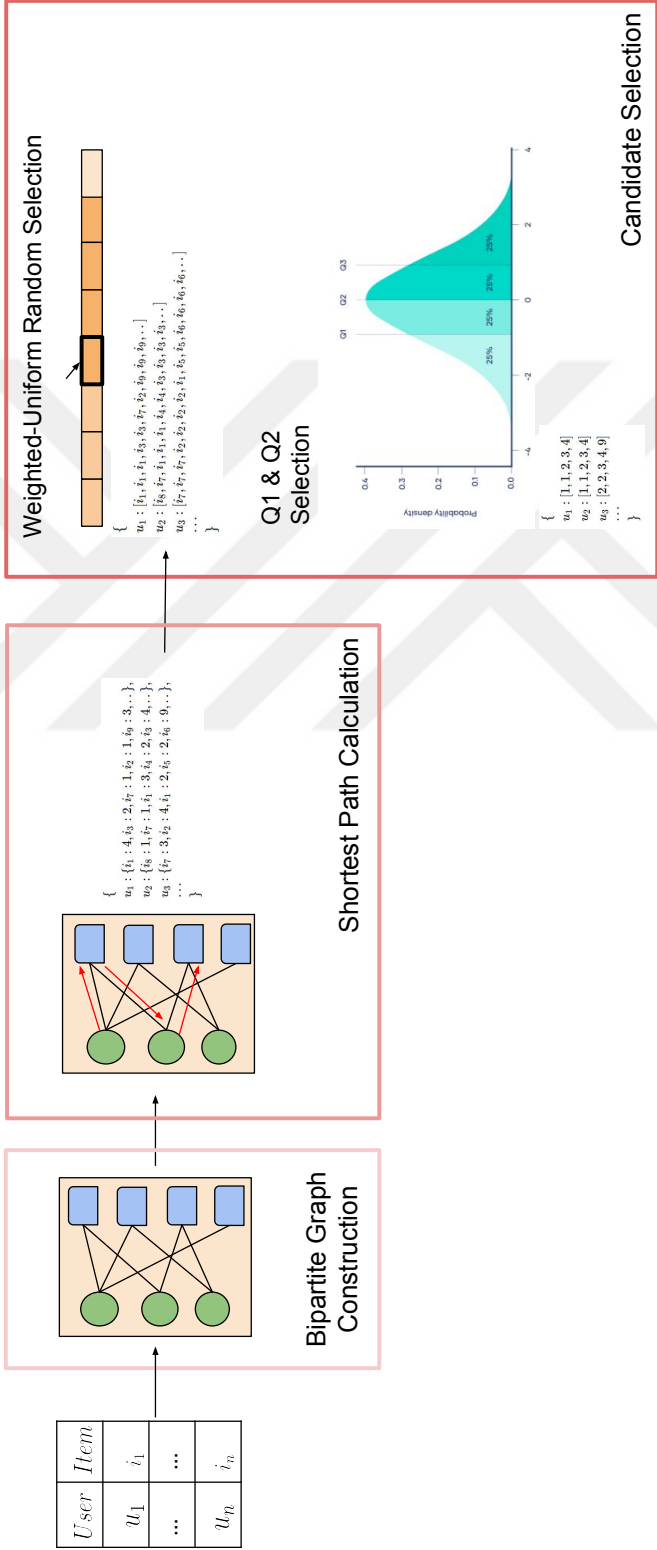


FIGURE 4.1 – Path-Length Negative Sampling Flow Chart

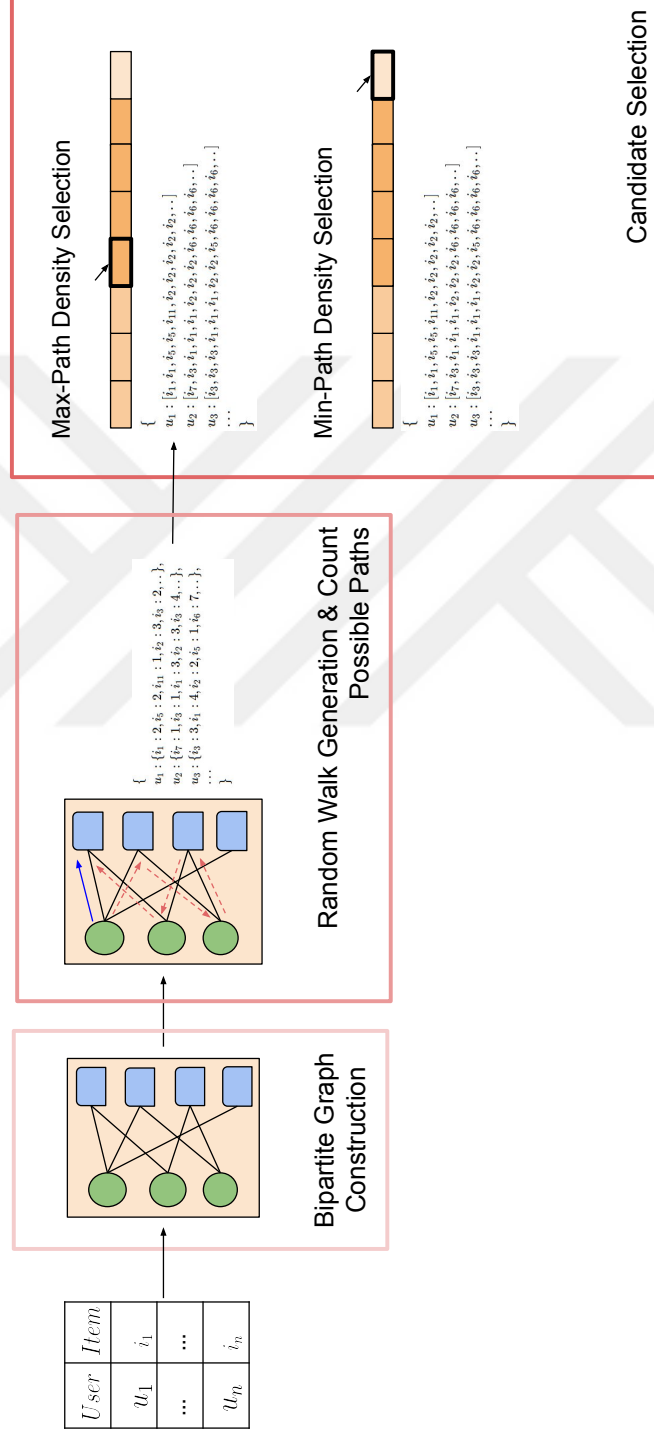


FIGURE 4.2 – Path-Density Negative Sampling Flow Chart

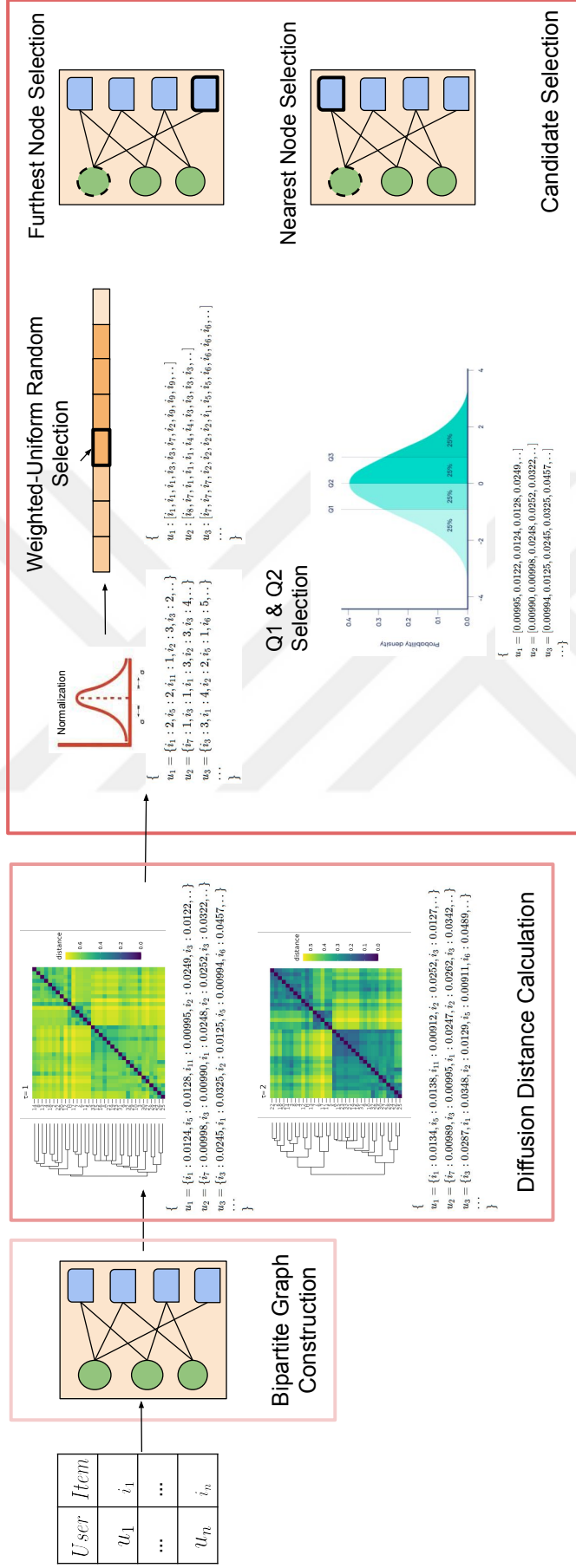


FIGURE 4.3 – Diffusion Distance Negative Sampling Flow Chart

(The images in the diffusion distance calculation process are courtesy from :

<https://gbertagnolli.github.io/diffudist/articles/diffusion-distances.html#references-1>)

## 5 EXPERIMENTS AND RESULTS

This chapter presents the experimental evaluation of two proposed solutions designed to address data sparsity and class imbalance issues faced by recommender systems. For our experiments, we utilized a dataset derived from the Last.fm online music system (*Last.fm website*, n.d.). This dataset, released within the framework of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011) (Cantador et al., 2011), captures various user interactions, including social networking, tagging, and music artist listening information for a collection of 1892 users. As detailed in Table 5.1, the dataset exhibits sparsity characteristics, meaning that users have interacted with a relatively small number of artists on average, which also leads the class imbalance issue. Having a sparse and class-imbalanced dataset is particularly well-suited for our research, as we aim to develop a solution that addresses these challenges in recommender systems.

TABLE 5.1 – Data Statistics of Last.fm dataset

| Data Category                         | Statistic |
|---------------------------------------|-----------|
| #Users                                | 1892      |
| #Friend Relations                     | 12717     |
| Average Friend Connections per User   | 13.443    |
| #Artists                              | 17632     |
| #User-Artist Interactions             | 92834     |
| Average Artists per User              | 49.067    |
| Average Users per Artist              | 5.265     |
| #Artists' Description Tags (Unique)   | 11946     |
| #Times users assigned tags to artists | 186479    |
| Average Tag Assignments per User      | 98.562    |
| Average Tags Received per Artist      | 14.891    |
| Average Distinct Tags per User        | 18.93     |
| Average Distinct Tags per Artist      | 8.764     |
| Sparsity                              | 99.72%    |

In the first section of this chapter, we will provide you the results of our proposed experimental design to mitigate the negative impacts of data sparsity.

To evaluate our proposed data granularity framework, we selected NGCF (Wang et al.,

2019), LightGCN (He et al., 2020), and SGL (Wu et al., 2021) for our experiments due to their demonstrated effectiveness in leveraging graph-based structures to enhance recommendation quality. NGCF integrates user-item interactions within a GCN framework, enabling effective propagation of collaborative signals. LightGCN simplifies the GCN architecture, focusing on essential propagation steps and thereby enhancing both efficiency and performance. SGL introduces a self-supervised learning mechanism that can potentially improve robustness to noise and sparsity. These models represent a diverse set of strategies for incorporating graph-based information into recommender systems, making them suitable for evaluating our proposed solutions to data sparsity and class imbalance.

Throughout the experiments, we followed the original implementations of each model provided by the comprehensive Python library, RecBole (Xu et al., 2022). All models utilize RNS with one sample per training instance by default.

We used the default hyper-parameter settings for the mentioned models in the Recbole library. These are as follows :

- NGCF utilized an embedding size of 64 dimensions to represent both users and items. The model employs three hidden layers, each with 64 dimensions, for message propagation. During training, no dropout was applied to nodes (`node_dropout = 0.0`), while a 10% dropout rate (`message_dropout = 0.1`) was used for edges. Finally, an L2 regularization weight of  $1e-5$  (`reg_weight`) was applied to penalize large weights in the model.
- LightGCN model also used an embedding size of 64 dimensions for users and items. It employed three message propagation layers (`n_layers = 3`) to capture user-item relationships. Similar to NGCF, L2 regularization with a weight of  $1e-5$  (`reg_weight`) was used to prevent overfitting.
- SGL model was configured to use edge dropout (`'type' = 'ED'`). It employed three layers (`n_layers = 3`) for message propagation. The temperature parameter for the softmax function in the self-supervised learning objective was set to 0.5 (`ssl_tau`). As with the other models, the embedding size was set to 64 dimensions. A dropout rate of 10% (`drop_ratio = 0.1`) was used. L2 regularization with a weight of  $1e-5$  (`reg_weight`) was applied, and the weight of the self-supervised learning objective in the overall loss function was set to 0.05 (`ssl_weight`).

This section presents a comparative analysis of our proposed negative sampling approaches against two common negative sampling techniques : RNS and PNS. These baselines were chosen due to their widespread use and simplicity. In Chapter 4, we have already explained our proposed techniques in detail.

To ensure a consistent evaluation, we applied RNS, and PNS as baseline negative sampling methods, and examined the effectiveness of our proposed negative sampling approaches with the LightGCN model (He et al., 2020). We strictly obey the LightGCN’s original implementation that utilized an embedding dimension of 64 with Xavier initialization for the embedding parameters (Glorot and Bengio, 2010). Adam (Kingma and Ba, 2014) was used as the optimizer with a default learning rate of 0.001 and a mini-batch size of 2048. The L2 regularization coefficient  $\lambda$  was set to 0.0001, and the number of layers (K) remained at the default value of 3. Our code is publicly available at [https://github.com/tripledoubleE/negative\\_sampling](https://github.com/tripledoubleE/negative_sampling).

To assess the effectiveness of our proposed negative sampling technique, we conducted experiments in three settings. First, evaluated our proposed negative sampling technique in isolation, without incorporating any RNS. Next, we explored a “quasi-random” combination, where for a randomly chosen positive instance for each user, we included one random negative sample alongside the results from our proposed method. This allows us to assess the impact of introducing randomness alongside our proposed technique. Finally, we investigated a “full random” combination, where we generated one random negative instance per positive instance and combined them with our proposed technique’s output. This scenario explores the influence of a higher degree of randomness while still utilizing our proposed method.

By presenting the results of these experiments, we aim to demonstrate the efficacy of our proposed solutions in improving the performance of recommender systems in sparse and class-imbalanced settings.

The experiments were conducted on a machine equipped with an NVIDIA Tesla P100-PCIE-12GB GPU and an Intel(R) Xeon(R) CPU E5-2697A v4 @ 2.60GHz. The implementation was carried out using Python 3.8 and RecBole 1.2.0.

### 5.1 Results for Data Sparsity Solution

In this section, we present the empirical results of our experimental design, evaluated on different levels of data granularity. We compare the performance of NGCF, LightGCN, and SGL models under various sampling performance metrics including Recall@10, Mean Reciprocal Rank@10 (MRR), Normalized Discounted Cumulative Gain@10 (NDCG), Hit@10, Precision@10, and Mean Average Precision@10 (MAP).

Firstly, we examine the sparsity of each data granularity level. As seen in Table 5.2, on one hand, data sparsity increases, when we add additional information to item representations, but on the other hand, data sparsity decreases when we apply the level projection to the data. For instance, although L1 data represents only user-item interactions, L1L4 is the projected form of L1. It includes additional nodes and edges encompassed from the most detailed level. By incorporating extra details, the model has a richer representation of user preferences and artist attributes, and we hypothesize that it may lead to better predictions.

The detailed results for each model (NGCF, LightGCN, and SGL) are presented in Tables 5.3, 5.4, and 5.5, respectively. Both NGCF, LightGCN and SGL exhibit signi-

ficant improvements in recommendation performance with the inclusion of additional user-artist interaction details (weights, tags, and temporal information) using our data augmentation strategy compared to using only user-artist interactions.

Table 5.3 shows the results of NGCF for each granularity level. The results demonstrate a clear improvement in all metrics as the granularity of the data increases. At L1, the performance is significantly lower across all metrics (Precision@10 : 0.0003, Recall@10 : 0.0027, NDCG@10 : 0.0009). The performance improves markedly at L2, and peaks at L3 with substantial increases (Precision@10 : 0.0927, Recall@10 : 0.9267, NDCG@10 : 0.7544)

The NGCF model achieves the highest performance in L3L4 for all performance metrics. This shows that our data projection strategy provides a better learning environment for the NGCF model by reducing the sparse data environment. In addition, we observe

a significant improvement in NDCG@10 and MRR, indicating that the model not only retrieves relevant artists but also prioritizes highly relevant ones among the best suggestions. The performance improvements from L1L4 to L3L4 show that the additional information provided at higher data granularity levels (weight and tag information) helps to improve the recommendation quality of the NGCF model.

On the other hand, including month/year information in L4 data seems to have a negative impact compared to L3L4. The key reasons for that may be 2-fold : (1) As seen in Table 5.2, incorporating time in L4 increases sparsity, adversely affecting NGCF’s performance. (2) User preferences for music might not strictly follow time. Including time detail might introduce irrelevant noise that the model needs to learn around.

TABLE 5.2 – Data Sparsity of created data granularity levels

| Data Granularity | Data Sparsity |
|------------------|---------------|
| L1               | 99.72%        |
| L2               | 99.95%        |
| L3               | 99.74%        |
| L4               | 99.91%        |
| L1L4             | 96.39%        |
| L2L4             | 99.4%         |
| L3L4             | 99.33%        |

TABLE 5.3 – Results for NGCF in Last.fm dataset for each data granularity

| Data Granularity | Precision@10  | Recall@10     | NDCG@10       | HR            | MRR           | MAP           |
|------------------|---------------|---------------|---------------|---------------|---------------|---------------|
| L1               | 0.0003        | 0.0027        | 0.0009        | 0.0027        | 0.0004        | 0.0004        |
| L2               | 0.0025        | 0.0246        | 0.0099        | 0.0246        | 0.0056        | 0.0056        |
| <b>L3</b>        | <b>0.0927</b> | <b>0.9267</b> | <b>0.7544</b> | <b>0.9267</b> | <b>0.6957</b> | <b>0.6957</b> |
| L4               | 0.0920        | 0.9206        | 0.7918        | 0.9206        | 0.7493        | 0.7493        |
| L1L4             | 0.0045        | 0.0454        | 0.0326        | 0.0454        | 0.0284        | 0.0284        |
| L2L4             | 0.0909        | 0.9096        | 0.8321        | 0.9096        | 0.8063        | 0.8063        |
| <b>L3L4</b>      | <b>0.0934</b> | <b>0.9342</b> | <b>0.8511</b> | <b>0.9342</b> | <b>0.8226</b> | <b>0.8226</b> |

Table 5.4 presents the results of LightGCN for each granularity level. We observe that the LightGCN shows a similar trend to the NGCF, with low performance at L1 (Precision@10 : 0.0004, Recall@10 : 0.0037, NDCG@10 : 0.0014) and a significant improvement as granularity increases. The highest performance is observed at L3 and L4, with L4 achieving slightly better results in terms of Precision@10 (0.0928) and Recall@10

(0.9280), but L3 shows the highest NDCG@10 (0.8529). The projected levels L3L4 exhibit the best overall performance in NDCG@10 (0.8656) and MRR (0.8560), suggesting that using level projection which decreases the data sparsity, leads to capturing diverse user-item interactions more effectively.

The choice between L3L4 and L4 depends on the specific needs. If precise ranking within the top 10 is crucial, L3L4 might be preferable. However, if maximizing the overall number of relevant recommendations and capturing the most relevant artists for users is the priority, L4 seems to be the most effective choice based on these results.

The margin between NGCF and LightGCN is relatively small for most metrics, suggesting they are both strong contenders for this task. Choosing the best model might depend on factors like computational efficiency or the specific needs of the recommender system. While L3L4 performs well for NGCF and L4 performs well for LightGCN, it's important to consider the trade-off between including more information and potential noise, since the increased complexity of higher data granularity levels may lead to longer training times.

TABLE 5.4 – Results for LighGCN in Last.fm dataset for each data granularity

| Data Granularity | Precision@10  | Recall@10     | NDCG@10       | HR            | MRR           | MAP           |
|------------------|---------------|---------------|---------------|---------------|---------------|---------------|
| L1               | 0.0004        | 0.0037        | 0.0014        | 0.0037        | 0.0007        | 0.0007        |
| L2               | 0.0021        | 0.0213        | 0.0079        | 0.0213        | 0.0040        | 0.0040        |
| <b>L3</b>        | 0.0912        | 0.9121        | <b>0.8529</b> | 0.9121        | <b>0.8325</b> | <b>0.8325</b> |
| <b>L4</b>        | <b>0.0928</b> | <b>0.9280</b> | 0.8393        | <b>0.9280</b> | 0.8094        | 0.8094        |
| L1L4             | 0.0033        | 0.0331        | 0.0271        | 0.0331        | 0.0251        | 0.0251        |
| L2L4             | 0.0891        | 0.8915        | 0.8142        | 0.8915        | 0.7878        | 0.7878        |
| <b>L3L4</b>      | 0.0892        | 0.8929        | <b>0.8656</b> | 0.8929        | <b>0.8560</b> | <b>0.8561</b> |

Table 5.5 displays the results of SGL on Last.fm data for each granularity level. SGL follows the similar pattern of improvement compared to NGCF and LightGCN with increasing granularity. At L1, the performance is minimal (Precision@10 : 0.0002, Recall@10 : 0.0021, NDCG@10 : 0.0007). The performance peaks at L3 (Precision@10 : 0.0786, Recall@10 : 0.7857, NDCG@10 : 0.5711) but slightly drops at L4. The increased sparsity at higher granularity levels at L4 with temporal data can exacerbate the model's difficulty in learning robust representations. SGL might lack the mechanisms to effectively integrate and leverage the additional contextual information, unlike NGCF and LightGCN, which may have more sophisticated propagation

and aggregation mechanisms better suited to handle complex, high-dimensional data. As a result, SGL constantly performs worse than the other two models at all data granularities. The results indicate that while NGCF and LightGCN can manage and even benefit from the additional information, SGL’s current architecture and learning approach may not be as well-suited for these more detailed data representations.

Additionally, the projected level L2L4 provides the best performance across all metrics (Precision@10 : 0.0825, Recall@10 : 0.8253, NDCG@10 : 0.6701). Unlike the LightGCN and NGCF models, SGL’s performance declines after L2L4. Including tag information (L3L4) and time data (L4) seems to hurt the model’s performance in this case. One technical reason for this could be the nature of SGL’s self-supervised learning objective, which relies heavily on contrastive learning. In contrastive learning, the model distinguishes between positive and negative samples by maximizing the agreement between positive pairs and minimizing it for negative pairs. The added complexity from tag and time information may introduce additional noise and less meaningful distinctions in the latent space, making it harder for the model to effectively differentiate between positive and negative samples.

Furthermore, SGL’s use of edge dropout for self-supervised learning might interact poorly with the increased density and heterogeneity of the data introduced at higher granularity levels. The edge dropout technique is designed to help the model generalize by randomly dropping edges during training. However, when combined with noisy or overly complex data, this could lead to a degradation in the quality of the learned embeddings.

TABLE 5.5 – Results for SGL in Last.fm dataset for each data granularity

| Data Granularity | Precision@10  | Recall@10     | NDCG@10       | HR            | MRR           | MAP           |
|------------------|---------------|---------------|---------------|---------------|---------------|---------------|
| L1               | 0.0002        | 0.0021        | 0.0007        | 0.0021        | 0.0003        | 0.0003        |
| L2               | 0.0020        | 0.0197        | 0.0074        | 0.0197        | 0.0040        | 0.0040        |
| <b>L3</b>        | <b>0.0786</b> | <b>0.7857</b> | <b>0.5711</b> | <b>0.7857</b> | <b>0.5046</b> | <b>0.5046</b> |
| L4               | 0.0620        | 0.6201        | 0.4257        | 0.6201        | 0.3648        | 0.3648        |
| L1L4             | 0.0031        | 0.0310        | 0.0227        | 0.0310        | 0.0201        | 0.0201        |
| <b>L2L4</b>      | <b>0.0825</b> | <b>0.8253</b> | <b>0.6701</b> | <b>0.8253</b> | <b>0.6227</b> | <b>0.6229</b> |
| L3L4             | 0.0707        | 0.7072        | 0.5788        | 0.7072        | 0.5394        | 0.5394        |

In summary, the key to improved performance across all models lies in our proposed experimental data granularity framework. In most cases, using more granular data

(e.g. L3, L4) created by the data augmentation strategy that we proposed leads to better performance compared to less granular data (e.g. L1). Projecting information from lower granular levels (L1) to higher granular levels (L1L4, L2L4, L3L4) fills in missing interactions and strengthens existing ones, reducing sparsity. For instance, if a user frequently interacts with artists tagged as “Rock” and “Alternative”, projecting this tag information to the user-artist interaction level (L2L4) can create implicit interactions even if the user hasn’t explicitly interacted with all Rock or Alternative artists. Moreover, additional information like weights and tags provide a richer representation of user preferences and artist attributes. Finally, by mitigating the negative effects of sparsity and enriching the user-artist representation, the M-GraP data granularity framework leads to more accurate recommendations.

## 5.2 Results for Class Imbalance Solution

This section presents the results of our experiments for the LightGCN method on the Last.fm dataset to evaluate the effectiveness of different path-length-based, path-density-based, and diffusion distance-based negative sampling approaches for recommender systems. We compared five negative sampling techniques : RNS, PNS, Path-Length-based negative sampling (PL-NS), Path-Density-based negative sampling (PD-NS), and Diffusion Distance-based Negative sampling (DD-NS). To ensure computational efficiency, we limited positive training examples to a maximum of 10 per user. If the user has more than 10 interactions, we choose randomly 10 of them. In addition, we take RNS as a baseline approach, where both 1 positive and 1 negative items were chosen randomly for each user. To evaluate performance, we used three performance evaluation metrics : Precision@10, Recall@10, and NDCG@10.

Table 5.6 presents the results of the Path-Length NS approach on the Last.fm dataset.

This approach is based on the shortest path between user and item nodes. The results are reported for different PL-NS strategies, including path-length long-distance (PL-Long-Distance), path-length Q1 (PL-Q1), and path-length Q1 (PL-Q2). To provide a basis for comparison, we include the outcomes achieved with RNS and PNS. This table also compares the performance of combining Full RNS (choosing one random negative instance for each positive instance) with path-length-based approaches (Full RNS +

PL), combining RNS with path-length-based approaches (RNS + PL), and using path-length-based approaches only (PL). It is shown that incorporating Full RNS + PL approaches does not necessarily improve the results. In most cases, using only PL achieves better or similar performance compared to Full RNS + PL and RNS + PL. This suggests that using path-length information alone might be sufficient to capture the negative sampling needs in this dataset.

The results demonstrate that using solely PL-Long-Distance or combining with RNS outperforms RNS and PNS on all evaluation metrics (Precision@10, Recall@10, and NDCG@10) in most configurations. For example, PL-Long-Distance with 1 negative item per positive item and 446 epochs achieves the best Precision@10 (0.0828), Recall@10 (0.1498), and NDCG@10 (0.1368) compared to RNS (0.0811, 0.1483, 0.1347) with the same configuration.

It is worth noting that while combining PL results with RNS, increasing negative examples per positive item doesn't always lead to better performance (as seen comparing rows with 1 vs 2 sampling ratio). However, increasing negatives from 1 to 2 yields the best outcome for PL-Long-Distance alone.

TABLE 5.6 – Path-Length NS Approach Results in Last.fm dataset

| Sampling Approach                  | #Neg Item<br>per Pos Item | Epoch      | Precision@10  | Recall@10     | NDCG@10       |
|------------------------------------|---------------------------|------------|---------------|---------------|---------------|
| RNS                                | 1                         | 999        | 0.0811        | 0.1483        | 0.1347        |
| PNS                                | 1                         | 999        | 0.079         | 0.1435        | 0.1295        |
| <b>Full RNS + PL-Long-Distance</b> | 1                         | <b>68</b>  | <b>0.0814</b> | 0.1474        | 0.1344        |
| <b>Full RNS + PL-Q1</b>            | 1                         | <b>61</b>  | <b>0.0812</b> | 0.1479        | 0.1335        |
| Full RNS + PL-Q2                   | 1                         | 61         | 0.08          | 0.1438        | 0.131         |
| <b>RNS + PL-Long-Distance</b>      | 1                         | <b>446</b> | <b>0.0827</b> | <b>0.1511</b> | <b>0.1361</b> |
| <b>RNS + PL-Q1</b>                 | 1                         | <b>262</b> | 0.0805        | 0.1481        | <b>0.1351</b> |
| RNS + PL-Q2                        | 1                         | 262        | 0.0773        | 0.1398        | 0.1277        |
| RNS + PL-Long-Distance             | 2                         | 438        | 0.0809        | 0.1465        | 0.1334        |
| <b>RNS + PL-Q1</b>                 | 2                         | <b>262</b> | 0.0805        | 0.1481        | <b>0.1351</b> |
| RNS + PL-Q2                        | 2                         | 263        | 0.0773        | 0.1398        | 0.1277        |
| <b>PL-Long-Distance</b>            | 1                         | <b>312</b> | <b>0.0825</b> | <b>0.1496</b> | 0.1351        |
| PL-Q1                              | 1                         | 184        | 0.0797        | 0.145         | 0.1324        |
| PL-Q2                              | 1                         | 183        | 0.0771        | 0.1379        | 0.125         |
| <b>PL-Long-Distance</b>            | 2                         | <b>304</b> | <b>0.0828</b> | <b>0.1498</b> | <b>0.1368</b> |
| PL-Q1                              | 2                         | 183        | 0.0797        | 0.145         | 0.1324        |
| PL-Q2                              | 2                         | 183        | 0.0771        | 0.1379        | 0.125         |

Table 5.7 summarizes the results of the Path Density NS approach on the Last.fm dataset. This approach calculates negative instances based on path density, including maximum and minimum strategies.

The results show that path-density-based negative sampling approaches underperform both RNS and PNS. PD-Min achieves a Precision@10 of 0.0804, which is close to RNS (0.0811), but lower Recall@10 (0.1437 vs 0.1483) and NDCG@10 (0.1314 vs 0.1347). This suggests it might identify some relevant negative examples, but misses others. The significant drop in performance with PD-Max (Precision@10 drops to 0.0667) is likely because it prioritizes selecting items with high path density, essentially focusing on items with more possible connections to the user. However, the fact that the results are bad indicates that these highly connected items are not more descriptive negative examples. This makes sense because if the path density between a user and an item is high, that item is more likely to be a positive item. Overall, these findings suggest that path density might not be a strong indicator for selecting effective negative items in this dataset.

TABLE 5.7 – Path-Density NS Approach Results in Last.fm dataset

| Sampling Approach | Epoch | Precision@10  | Recall@10     | NDCG@10       |
|-------------------|-------|---------------|---------------|---------------|
| <b>RNS</b>        | 999   | <b>0.0811</b> | <b>0.1483</b> | <b>0.1347</b> |
| PNS               | 999   | 0.079         | 0.1435        | 0.1295        |
| PD-Max            | 800   | 0.0667        | 0.1237        | 0.0186        |
| PD-Min            | 105   | 0.0804        | 0.1437        | 0.1314        |

Table 5.8 presents the results of various strategies used DD-NS in Last.fm dataset.

This approach is based on the diffusion distance between user and item nodes. We first examine using the farthest and nearest items based on their diffusion distance as negative items for users as a starting point. It’s worth noting the two key points : First using the nearest node significantly hurts the model’s performance, this makes sense because nodes that are closer to the user in the network are more likely to be items the user has interacted with or might be interested in, making them the less effective negative example. Second both the nearest and farthest node strategies perform much worse than other approaches. While selecting the farthest node might seem like a good way to find negative examples, the results suggest these distant nodes might not provide enough context or relevance for the model. This reinforces the idea that an effective

negative sampling strategy should target nodes that are neither too close nor too far from the user in the network.

In addition, we can see that, the Q1 and Q2 selection strategies consistently underperform compared to the various “Long-Distance” approaches, they perform like the nearest strategy. We observed that the “Long-Distance” approaches consistently outperformed the RNS baseline and the other strategies across all evaluation metrics (Precision@10, Recall@10, NDCG@10) in both Table 5.8 and Table 5.9. The results suggest

that incorporating artists farther away from users in the recommendation process, with weights based on distance and followed by random selection, improves recommendation quality. Compared to the baseline RNS approach (Precision@10 : 0.0811, Recall@10 : 0.1483, NDCG@10 : 0.1347), DD-NS-Long-Distance achieves significant improvement in all metrics (e.g., Precision@10 : 0.0823 to 0.0836, Recall@10 : 0.1489 to 0.1536, NDCG@10 : 0.1363 to 0.1386).

Moreover, considering the Long-Distance strategies, we applied normalization on the diffusion distances (detailed in subsection 4.2.3) and investigated the impact of normalization on long-distance strategies, we scaled the data to three different ranges : [1, 5], [1, 10], and [1, 100]. We then evaluated the effect of each normalization scheme on the model’s performance. While the best-performing approach (DD-NS-Long-Distance (scale : [1,5],  $\tau = 0.5$ )) remains consistent for both as in Table 5.8 and in Table 5.9

there were slight variations in scores between scales. These differences are small and don’t exhibit a discernible pattern. This would result in the application of comparable weightings irrespective of the initial scale.

TABLE 5.8 – DD-NS Results in Last.fm dataset

| Sampling Approach                     | Epoch | tau | Precision@10  | Recall@10     | NDCG@10       |
|---------------------------------------|-------|-----|---------------|---------------|---------------|
| RNS                                   | 999   | -   | 0.0811        | 0.1483        | 0.1347        |
| PNS                                   | 999   | -   | 0.079         | 0.1435        | 0.1295        |
| DD-NS-Farthest                        | 163   | 1   | 0.0109        | 0.0234        | 0.0175        |
| DD-NS-Nearest                         | 163   | 1   | 0.0161        | 0.0318        | 0.0255        |
| DD-NS-Q1                              | 266   | 0.5 | 0.0145        | 0.0279        | 0.0216        |
| DD-NS-Q1                              | 268   | 1   | 0.0231        | 0.042         | 0.0335        |
| DD-NS-Q1                              | 266   | 2   | 0.0323        | 0.0594        | 0.0489        |
| DD-NS-Q1                              | 266   | 4   | 0.0379        | 0.0673        | 0.0552        |
| DD-NS-Q2                              | 265   | 0.5 | 0.0111        | 0.0231        | 0.0179        |
| DD-NS-Q2                              | 264   | 1   | 0.0112        | 0.0233        | 0.018         |
| DD-NS-Q2                              | 266   | 2   | 0.0157        | 0.031         | 0.0232        |
| DD-NS-Q2                              | 266   | 4   | 0.0375        | 0.0674        | 0.056         |
| DD-NS-Long-Distance (scale : [1,5])   | 250   | 0.5 | <b>0.083</b>  | <b>0.1515</b> | <b>0.1382</b> |
| DD-NS-Long-Distance (scale : [1,5])   | 218   | 1   | 0.0809        | 0.1474        | <b>0.1351</b> |
| DD-NS-Long-Distance (scale : [1,5])   | 250   | 2   | <b>0.0822</b> | <b>0.1492</b> | <b>0.1364</b> |
| DD-NS-Long-Distance (scale : [1,5])   | 250   | 4   | <b>0.0824</b> | <b>0.1502</b> | <b>0.1367</b> |
| DD-NS-Long-Distance (scale : [1,10])  | 250   | 0.5 | 0.0811        | 0.1477        | <b>0.1363</b> |
| DD-NS-Long-Distance (scale : [1,10])  | 245   | 1   | <b>0.0827</b> | <b>0.1505</b> | <b>0.1363</b> |
| DD-NS-Long-Distance (scale : [1,10])  | 250   | 2   | <b>0.0827</b> | <b>0.1494</b> | <b>0.1362</b> |
| DD-NS-Long-Distance (scale : [1,10])  | 250   | 4   | <b>0.0815</b> | <b>0.1481</b> | <b>0.1348</b> |
| DD-NS-Long-Distance (scale : [1,100]) | 250   | 0.5 | <b>0.0823</b> | <b>0.1489</b> | <b>0.1368</b> |
| DD-NS-Long-Distance (scale : [1,100]) | 250   | 1   | <b>0.0825</b> | <b>0.1483</b> | <b>0.1363</b> |
| DD-NS-Long-Distance (scale : [1,100]) | 250   | 2   | <b>0.0814</b> | 0.1467        | 0.1345        |
| DD-NS-Long-Distance (scale : [1,100]) | 250   | 4   | <b>0.0829</b> | <b>0.15</b>   | <b>0.1369</b> |

Table 5.9 presents the results of various strategies using the combined negative sampling results from RNS and DD-NS approaches in Last.fm dataset. As in Table 5.8, we first examine how using the farthest and nearest items based on their diffusion distance to the user performs as a starting point. Their performance is similar to each other and significantly lower than other approaches. We can see that although the Q1 and Q2 selection strategies show lower performance compared to the different “Long-Distance” strategies, they perform better compared to solely using the DD-NS approach in Table 5.8. The reason for that is, adding the random negative samples, and we understand that the randomness has a positive effect in this case.

Similar to Table 5.8, the best-performing approach is consistently RNS + DD-NS-Long-Distance across all configurations. We observed that combining RNS with DD-NS might lead to a small improvement in retrieving all relevant items for users.

TABLE 5.9 – RNS + DD-NS Results in Last.fm dataset

| Sampling Approach                            | Epoch | tau | Precision@10  | Recall@10     | NDCG@10       |
|----------------------------------------------|-------|-----|---------------|---------------|---------------|
| RNS                                          | 999   | -   | 0.0811        | 0.1483        | 0.1347        |
| PNS                                          | 999   | -   | 0.079         | 0.1435        | 0.1295        |
| RNS + DD-NS-Farthest                         | 162   | 1   | 0.0752        | 0.1355        | 0.1207        |
| RNS + DD-NS-Nearest                          | 162   | 1   | 0.0549        | 0.1001        | 0.0825        |
| RNS + DD-NS-Q1                               | 265   | 0.5 | 0.0774        | 0.1372        | 0.1217        |
| RNS + DD-NS-Q1                               | 261   | 1   | 0.0759        | 0.1365        | 0.1227        |
| RNS + DD-NS-Q1                               | 264   | 2   | 0.0733        | 0.1328        | 0.1193        |
| RNS + DD-NS-Q1                               | 264   | 4   | 0.0788        | 0.1416        | 0.1277        |
| RNS + DD-NS-Q2                               | 263   | 0.5 | 0.0786        | 0.1417        | 0.1272        |
| RNS + DD-NS-Q2                               | 263   | 1   | 0.0765        | 0.1389        | 0.1248        |
| RNS + DD-NS-Q2                               | 262   | 2   | 0.0731        | 0.132         | 0.118         |
| RNS + DD-NS-Q2                               | 260   | 4   | 0.0761        | 0.1396        | 0.1258        |
| RNS + DD-NS-Long-Distance (scaled : [1-5])   | 250   | 0.5 | <b>0.0836</b> | <b>0.1536</b> | <b>0.1386</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-5])   | 218   | 1   | <b>0.0821</b> | <b>0.1511</b> | <b>0.1362</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-5])   | 250   | 2   | <b>0.0822</b> | <b>0.1492</b> | <b>0.1363</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-5])   | 250   | 4   | <b>0.0821</b> | <b>0.1502</b> | <b>0.1378</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-10])  | 250   | 0.5 | 0.0806        | 0.1465        | 0.1346        |
| RNS + DD-NS-Long-Distance (scaled : [1-10])  | 250   | 1   | <b>0.0826</b> | <b>0.1486</b> | <b>0.1363</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-10])  | 250   | 2   | <b>0.0824</b> | <b>0.1496</b> | <b>0.1357</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-10])  | 250   | 4   | <b>0.0823</b> | <b>0.1494</b> | <b>0.1364</b> |
| RNS + DD-NS Long-Distance (scaled : [1-100]) | 250   | 0.5 | <b>0.0815</b> | <b>0.1485</b> | <b>0.1354</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-100]) | 250   | 1   | <b>0.0823</b> | <b>0.1491</b> | <b>0.1354</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-100]) | 250   | 2   | <b>0.082</b>  | <b>0.1508</b> | <b>0.1368</b> |
| RNS + DD-NS-Long-Distance (scaled : [1-100]) | 250   | 4   | <b>0.0817</b> | 0.1482        | <b>0.1352</b> |

Furthermore, the DD-NS technique focused on the network’s diffusion process, which changes as time passes. The parameter  $\tau$  controls the diffusion time, essentially defining how far a random walk is allowed to explore the network. When tau is low, the diffusion process is limited. Nodes connected by direct edges or very short paths will have a low distance, while those further apart will have a high distance. This emphasizes local similarities. With higher tau, the diffusion process has more time to explore the network. Nodes further connected through longer paths will have a lower distance compared to the low tau case. This captures more distant similarities. In the context of DD-NS-Q1, Q2, and RNS + DD-NS-Q1, Q2 generally higher tau values lead to finding more relevant recommendations. This suggests that considering items further away in the network using a higher diffusion time (higher tau) might be beneficial for determining the negative items. Considering the Long-Distance strategy, the optimal tau value depends on the specific task. For instance, regarding DD-NS, for  $scale = [1, 5]$ , the optimal tau is 0.5, but for  $scale = [1, 100]$ , the optimal tau is 4.

Table 5.10 summarizes the best-performing configurations from our experiments on Last.fm dataset. “Long-Distance” techniques, either PL-Long-Distance or DD-NS-Long-Distance, seem to be effective for achieving the best performance. These techniques allow the model to learn better, possibly by finding the “actual” negatives in the network in a field-accurate way. Combining RNS with these “Long-Distance” approaches appears to be the most effective strategy in this dataset, achieving the highest Precision@10 and NDCG@10.

Building on the observation in (Yang et al., 2024) that Hard NS aids model training by providing clear gradient updates, we demonstrate that all proposed negative sampling approaches in this thesis lead to faster and more clear model convergence (as can be seen in Figure. 5.5).

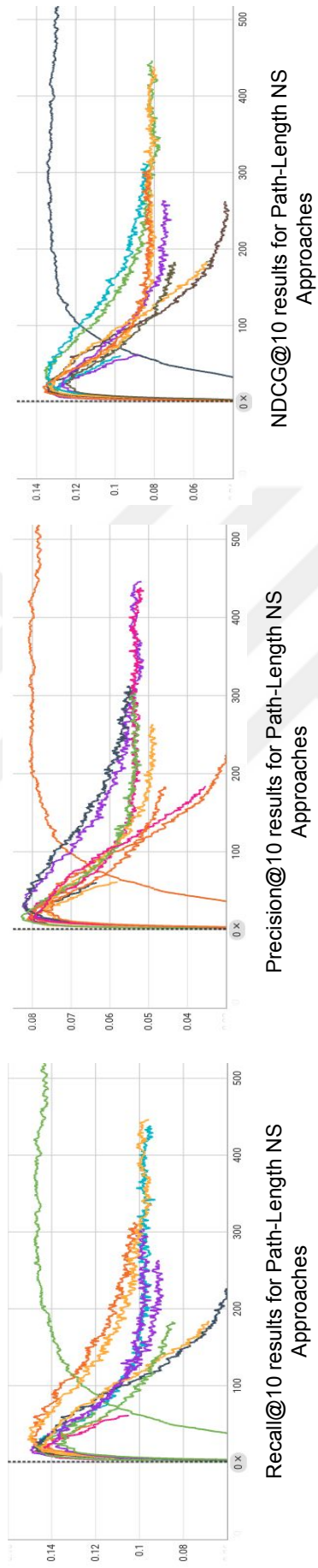


FIGURE 5.1 – Path-Length NS Approaches Results

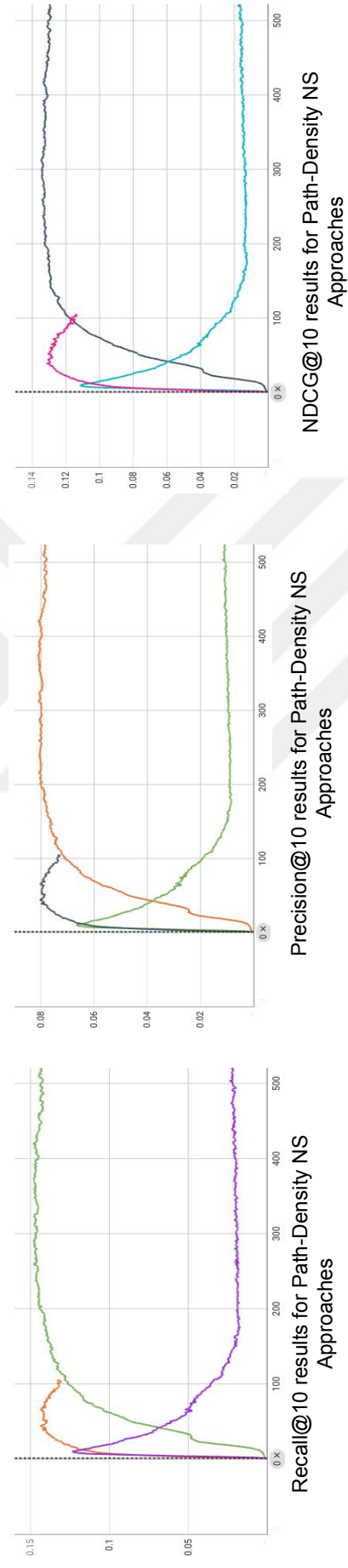


FIGURE 5.2 – Path-Length NS Approaches Results

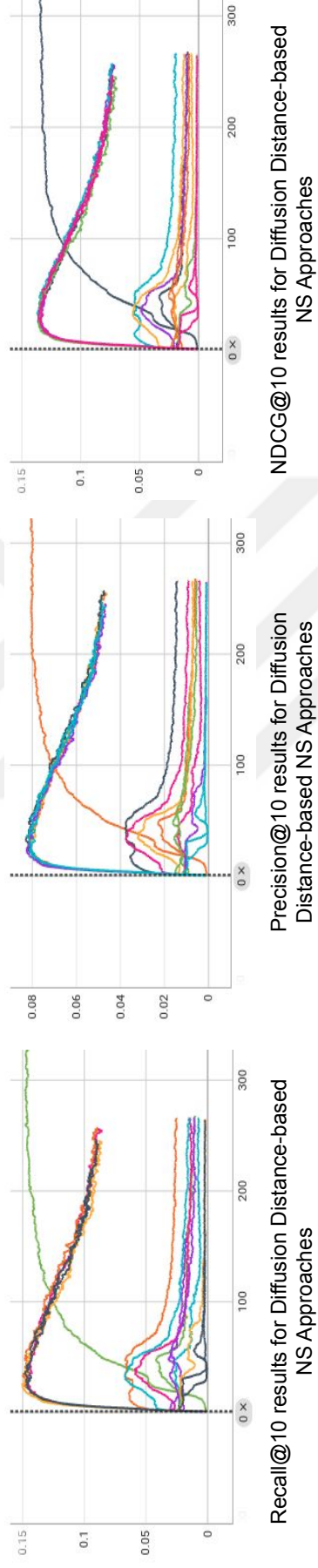


FIGURE 5.3 – Diffusion Distance-based NS Approaches Results

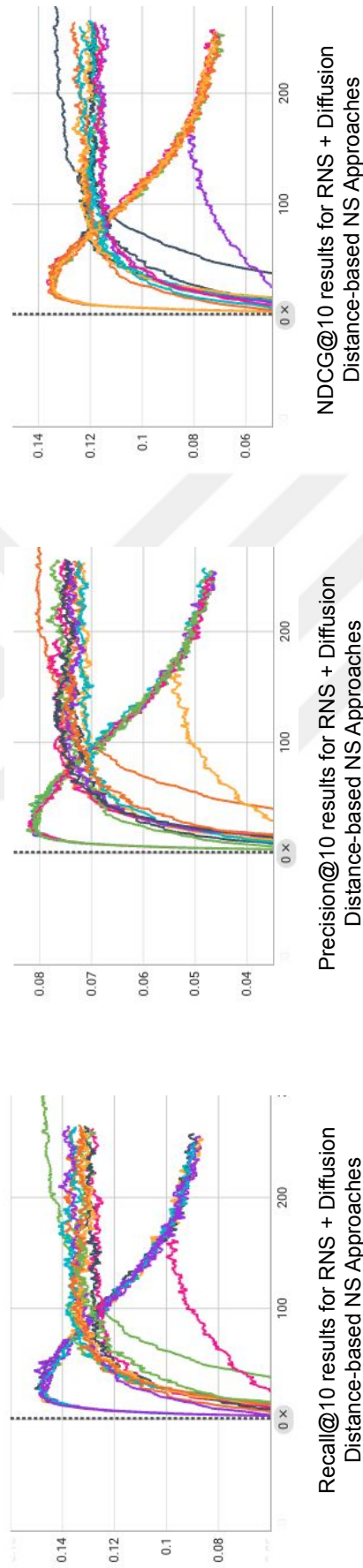


FIGURE 5.4 – RNS + Diffusion Distance-based NS Approaches Results

TABLE 5.10 – The best NS results in Last.fm dataset

| Sampling Approach                                 | #Neg Item<br>per Pos Item | Epoch      | tau        | Precision@10  | Recall@10     | NDCG@10       |
|---------------------------------------------------|---------------------------|------------|------------|---------------|---------------|---------------|
| RNS                                               | 1                         | 999        | -          | 0.0811        | 0.1483        | 0.1347        |
| PNS                                               | 1                         | 999        | -          | 0.079         | 0.1435        | 0.1295        |
| RNS + PL-Long-Distance                            | 1                         | 446        | -          | 0.0827        | 0.1511        | 0.1361        |
| PL-Long-Distance                                  | 2                         | 304        | -          | 0.0828        | 0.1498        | 0.1368        |
| Path Density Min                                  | 1                         | 105        | -          | 0.0804        | 0.1437        | 0.1314        |
| DD-NS-Long-Distance (scale : [1,5])               | 1                         | 250        | 0.5        | 0.083         | 0.1515        | 0.1382        |
| <b>RNS + DD-NS-Long-Distance (scaled : [1-5])</b> | <b>1</b>                  | <b>250</b> | <b>0.5</b> | <b>0.0836</b> | <b>0.1536</b> | <b>0.1386</b> |

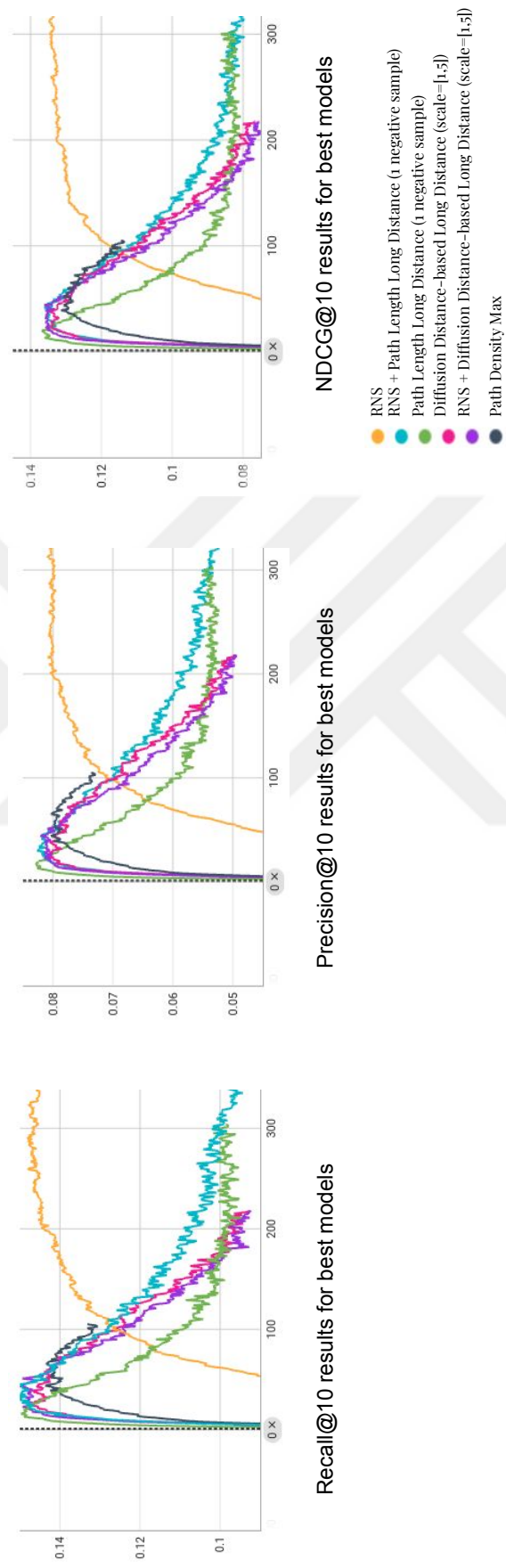


FIGURE 5.5 – The best results from each proposed negative sampling category

### 5.3 Discussion

In this section, we first go over the main conclusions of the suggested experimental design, which provides richer data representations for recommender systems using item attributes. We evaluated the performance of three models (NGCF, LightGCN, and SGL) on a music recommendation dataset with different levels of data granularity.

The experimental results underscore the importance of data granularity in recommender systems. Higher levels of granularity, even with sparse settings, significantly improve recommendation accuracy across all algorithms tested. Both NGCF, LightGCN, and SGL models achieved significant improvements in most evaluation metrics (Recall@10, MRR@10, NDCG@10, Hit@10, Precision@10, and MAP) when additional information like weights, tags, and temporal details were included. This enhancement can be attributed to the richer, more detailed representation of user-item interactions, allowing the models to capture nuances in user preferences and item characteristics.

While increasing granularity is generally associated with better performance, there is a threshold beyond which additional granularity may not significantly improve performance and may even degrade it. This is evident from the slight decrease in metrics at L4 compared to L3 for NGCF and LightGCN. Experimental work is needed to determine the optimal level of detail. Moreover, our experiments have shown that level projection, a strategy that reduces data sparsity, improves performance. In particular, projecting L3 to the L4 granular level appears to lead to the highest recommendation accuracy by providing the most comprehensive understanding of user behavior. This suggests that our data projection strategy, aimed at mitigating the negative effects of data sparsity, has enhanced the performance of the models.

For practitioners, these findings suggest that incorporating multiple levels of data granularity in recommender systems can mitigate the negative effects of data sparsity and enhance performance. However, the optimal level of granularity should be carefully determined to avoid unnecessary complexity and potential performance degradation. Moreover, projecting different granular levels can provide a more robust and effective recommendation system.

Secondly, we discuss the key findings from the experiments evaluating the effectiveness of our proposed network-science-oriented negative sampling approaches to address class imbalance issue. The Last.fm dataset was subjected to the LightGCN method using the following three evaluation metrics : Precision@10, Recall@10, and NDCG@10.

The results suggest that path-length information can be a valuable factor for negative sampling in recommender systems. PL-Long-Distance, which selects negative items with longer paths from the user node, outperformed both RNS and PNS on all evaluation metrics in most configurations. This suggests that selecting negative instances that are not directly connected to the user (via longer paths) may be more beneficial for training the model. Moreover, incorporating Full RNS with PL approaches did not necessarily improve the results. In most cases, utilizing merely PL resulted in better performance than Full RNS + PL. This shows that path-length information alone may be adequate to capture the negative sampling requirements in this dataset.

The findings for path-density-based negative sampling approaches were less promising. Both PD-Min and PD-Max underperformed compared to RNS and PNS. These findings indicate that path density may not be a reliable measure for identifying effective negative items in this dataset.

More promising results were obtained for DD-NS, which makes use of the diffusion distance between the user and item nodes. Diffusion distance-based selection of the farthest and nearest items underperformed other strategies by a large margin, indicating that these extremes are not optimal for negative sampling. DD-NS-Long-Distance was consistently the best-performing method; it identifies items in the network that are farther away from the user, weighted according to distance, and then selects them at random. When this method was used instead of the RNS baseline, all metrics showed notable improvements. There were minor differences in the scores when the normalization scheme for diffusion distances was changed, but there was no discernible pattern and the performance was not greatly impacted overall. Combining RNS with DD-NS-Long-Distance proved to be the most successful tactic. Comparing this method to utilizing DD-NS alone, users were able to retrieve relevant items with a slight improvement. This implies that a certain amount of unpredictability in negative sampling may be advantageous.

Our findings for all of our proposed negative sampling techniques showed that we can achieve the best performance quicker than the baseline method, RNS. It supports the (Yang et al., 2024) saying that Hard NS is an effective negative sampling method to speed up convergence and boost performance.

In addition to all these, all the negative sampling approaches we proposed within the scope of this thesis worked much slower than the RNS we used as a baseline. The reason for this is that for our proposed negative sampling approaches, we performed “global selection”. Moreover, the negative sampling methods we proposed enabled us to get better results on Last.fm data using the LightGCN model, but in order to make generalizations, we need to conduct more comprehensive experiments on both different models and different data.

Moreover, the computational efficiency of the proposed negative sampling techniques needs to be addressed. Optimizing these methods for faster execution while maintaining their effectiveness would be a valuable area for future research.

## 6 CONCLUSION

This thesis tackles the significant issues of data sparsity and class imbalance in recommendation systems. Our study presented new approaches that use item attributes and network topology information to improve recommendation accuracy.

The first major contribution of this thesis is to set the data granularity framework for CF with item attributes. In this line, first, we formalized the “data granularity concept” and presented a systematic methodology for data granularity that incorporates item characteristics for extracting optimal bipartite graph representation. This optimal bipartite graph can be used as the input to CF algorithms. By modeling user-item interactions as bipartite graphs at different levels of granularity, we enriched item representations and reduced the negative impacts of data sparsity.

The experimental results show that when compared to the least detailed level (L1) and its projected version to the most detailed level (L1L4), data sparsity was reduced from 99.72% to 96.39% using our data projection strategy. However, performance evaluations at this level (L1L4) showed low results due to noise from additional item nodes, indicating that sparsity is not always linearly correlated with performance enhancement. In addition, including more detailed item features (i.e. more granular representations) increased recommendation performance, while there was a proper granularity level for items. After this proper granularity level, more detailed representations had negative effects on the models. For instance, the NGCF model achieved the highest overall performance, with Precision@10 increased from 0.0045 to 0.0934, and the NDCG@10 improved from 0.0326 to 0.8511 compared to the proper granularity level. However, when we increase the data granularity one more level, the performance decreases in all metrics. For example for NDCG@10, it decreases by 6.9%.

In this thesis, we have demonstrated the effectiveness of our proposed data granularity framework, M-GraP, for optimal bipartite graph extraction through CF algorithms. Since the data granularity framework targets data representation, it can be used for other purposes different from the recommendation task or improving the success of CF

algorithms. Future research could expand data granularity experiments that project to every level and explore the application of the M-GraP framework in different domains and datasets to validate its generalizability. Additionally, investigating the trade-offs between computational complexity and performance benefits at various granular levels would provide deeper insights into the practical implementation of multi-granular recommender systems.

For the second contribution, we proposed several novel negative sampling approaches based on network science principles, including Path-Length, Path-Density, and Diffusion Distance. By focusing on the relative positions of items in the network, our methods identified more relevant negative examples, improving the training process and recommendation accuracy. Path-Length-based sampling, particularly the PL-Long-Distance approach increased the Precision@10 and Recall@10 by 1.9% and 2.8%, respectively, compared to traditional methods. In this view, the best refinement of this approach was seen in the RNS combined Diffusion Distance-based sampling, which recorded an increase in Recall@10 by 3% and NDCG@10 by 3.5%.

These network-based negative sampling strategies effectively reduced the negative effects of class imbalance, facilitating faster and more accurate model convergence. While these methods are independent, combining them when attributed data is available can provide a more robust solution to these challenges, further enhancing recommendation system performance and versatility.

Although the experiments in this thesis demonstrate the effectiveness of the proposed approaches, there are limitations. The proposed negative sampling strategies cause large processing costs since they examine the entire network to find the negative sample. Future work will investigate the use of local selection or mini-batch selection methodologies to alleviate these limitations and increase the scalability of comprehensive recommendation systems. Additionally, future research will focus on optimizing hyperparameters and evaluating these methodologies across diverse datasets to ensure generalizability and effectiveness in various domains, such as movie, hotel, and product recommendations.

In conclusion, this thesis has significantly addressed the difficulties of data sparsity and class imbalance in recommendation systems. We formalized the data granularity

problem to reduce data sparsity. We put forward novel approaches in using item attributes and network topology information for recommendation systems, and we proved that this showed remarkable improvement. These contributions will constitute the steps toward more accurate recommendations and better experiences for users in a wide range of applications.



## REFERENCES

- Abbasi-Moud, Z., Vahdat-Nejad, H. and Sadri, J. (2021). Tourism recommendation system based on semantic clustering and sentiment analysis, *Expert Systems with Applications* **167** : 114324.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0957417420310174>
- Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems : a survey of the state-of-the-art and possible extensions, *IEEE Transactions on Knowledge and Data Engineering* **17**(6) : 734–749.
- Bai, T., Wen, J.-R., Zhang, J. and Zhao, W. X. (2017). A neural collaborative filtering model with interaction-based neighborhood, *Proceedings of the ACM on CIKM*, p. 1979–1982.
- Beel, J., Gipp, B., Langer, S. and Breitingner, C. (2015). Research-paper recommender systems : A literature survey, *International Journal on Digital Libraries* pp. 1–34.
- Beel, J., Langer, S., Genzmehr, M., Gipp, B., Breitingner, C. and Nürnberger, A. (2013). Research paper recommender system evaluation : a quantitative literature survey, *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, RepSys '13, Association for Computing Machinery, New York, NY, USA, p. 15–22.  
**URL:** <https://doi.org/10.1145/2532508.2532512>
- Bertagnolli, G. and De Domenico, M. (2021). Diffusion geometry of multiplex and interdependent systems, *Phys. Rev. E* **103** : 042301.  
**URL:** <https://link.aps.org/doi/10.1103/PhysRevE.103.042301>
- Bobadilla, J., Ortega, F., Hernando, A. and Gutiérrez, A. (2013). Recommender systems survey, *Knowledge-Based Systems* **46** : 109–132.
- Cai, R., Wu, J., San, A., Wang, C. and Wang, H. (2021). Category-aware collaborative sequential recommendation, *Proceedings of the 44th International ACM SIGIR*

- Conference on Research and Development in Information Retrieval*, SIGIR '21, Association for Computing Machinery, New York, NY, USA, p. 388–397.  
**URL:** <https://doi.org/10.1145/3404835.3462832>
- Cantador, I., Brusilovsky, P. and Kuflik, T. (2011). 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011), *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, ACM, New York, NY, USA.
- Chen, L., Yuan, F., Jose, J. M. and Zhang, W. (2018). Improving negative sampling for word representation using self-embedded features, *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 99–107.  
**URL:** <https://doi.org/10.48550/arXiv.1710.09805>
- Chen, M., Li, Y. and Zhou, X. (2021). Conet : Co-occurrence neural networks for recommendation, *Future Gener. Comput. Syst.* **124**(C) : 308–314.  
**URL:** <https://doi.org/10.1016/j.future.2021.06.008>
- Chen, M. and Liu, P. (2017). Performance evaluation of recommender systems, *International Journal of Performability Engineering* **13**(8) : 1246.
- Chen, R., Hua, Q., Chang, Y.-S., Wang, B., Zhang, L. and Kong, X. (2018). A survey of collaborative filtering-based recommender systems : From traditional methods to hybrid methods based on social networks, *IEEE Access* **6** : 64301–64320.
- Chen, T., Kornblith, S., Norouzi, M. and Hinton, G. (2020). A simple framework for contrastive learning of visual representations, *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, JMLR.org.
- Chiny, M., Chihab, M., Bencharef, O. and Younes, C. (2022). Netflix recommendation system based on tf-idf and cosine similarity algorithms, pp. 15–20.
- Chuang, C.-Y., Robinson, J., Lin, Y.-C., Torralba, A. and Jegelka, S. (2020). Debiased contrastive learning, in H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin (eds), *Advances in Neural Information Processing Systems*, Vol. 33, Curran Associates, Inc., pp. 8765–8775.
- Cremonesi, P., Koren, Y. and Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks, RecSys '10, Association for Computing

Machinery, New York, NY, USA, p. 39–46.

**URL:** <https://doi.org/10.1145/1864708.1864721>

Da’u, A. and Salim, N. (2020). Recommendation system based on deep learning methods : a systematic review and new directions, *Artificial Intelligence Review* **53**(4) : 2709–2748.

**URL:** <https://doi.org/10.1007/s10462-019-09744-1>

Di Noia, T., Mirizzi, R., Ostuni, V. C., Romito, D. and Zanker, M. (2012). Linked open data to support content-based recommender systems, I-SEMANTICS ’12, Association for Computing Machinery, New York, NY, USA, p. 1–8.

**URL:** <https://doi.org/10.1145/2362499.2362501>

Ekstrand, M. D., Riedl, J. T. and Konstan, J. A. (2011). Collaborative filtering recommender systems, *Foundations and Trends® in Human-Computer Interaction* **4**(2) : 81–173.

**URL:** <http://dx.doi.org/10.1561/1100000009>

Fayyaz, Z., Ebrahimian, M., Nawara, D., Ibrahim, A. and Kashef, R. (2020). Recommendation systems : Algorithms, challenges, metrics, and business opportunities, *Applied Sciences* **10**(21).

Ferrari Dacrema, M., Boglio, S., Cremonesi, P. and Jannach, D. (2021). A troubling analysis of reproducibility and progress in recommender systems research, *ACM Trans. Inf. Syst.* **39**(2).

**URL:** <https://doi.org/10.1145/3434185>

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feed-forward neural networks, in Y. W. Teh and M. Titterton (eds), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Vol. 9 of *Proceedings of Machine Learning Research*, PMLR, Chia Laguna Resort, Sardinia, Italy, pp. 249–256.

**URL:** <https://proceedings.mlr.press/v9/glorot10a.html>

Grčar, M., Mladenič, D., Fortuna, B. and Grobelnik, M. (2006). Data sparsity issues in the collaborative filtering framework, in O. Nasraoui, O. Zaiane, M. Spiliopoulou, B. Mobasher, B. Masand and P. S. Yu (eds), *Advances in Web Mining and Web Usage Analysis*, Springer Berlin Heidelberg.

- Grover, A. and Leskovec, J. (2016). node2vec : Scalable feature learning for networks, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, Association for Computing Machinery, New York, NY, USA, p. 855–864.  
**URL:** <https://doi.org/10.1145/2939672.2939754>
- Guo, G. (2013). Integrating trust and similarity to ameliorate the data sparsity and cold start for recommender systems, RecSys '13, Association for Computing Machinery, New York, NY, USA, p. 451–454.  
**URL:** <https://doi.org/10.1145/2507157.2508071>
- Hamilton, W. L., Ying, R. and Leskovec, J. (2017). Inductive representation learning on large graphs, *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, p. 1025–1035.
- He, X., Deng, K., Wang, X., Li, Y., Zhang, Y. and Wang, M. (2020). Lightgcn : Simplifying and powering graph convolution network for recommendation, *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, Association for Computing Machinery, New York, NY, USA, p. 639–648.  
**URL:** <https://doi.org/10.1145/3397271.3401063>
- He, X., Liao, L., Zhang, H., Nie, L., Hu, X. and Chua, T.-S. (2017). Neural collaborative filtering, *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, p. 173–182.  
**URL:** <https://doi.org/10.1145/3038912.3052569>
- Hu, Y., Koren, Y. and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets, *2008 Eighth IEEE ICDM*, pp. 263–272.
- Huang, T., Dong, Y., Ding, M., Yang, Z., Feng, W., Wang, X. and Tang, J. (2021). Mixgcf : An improved training method for graph neural network-based recommender systems, *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, Association for Computing Machinery, New York, NY, USA, p. 665–674.  
**URL:** <https://doi.org/10.1145/3447548.3467408>

- Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques, *ACM Trans. Inf. Syst.* **20**(4) : 422–446.
- Jiang, Z., Chi, C. and Zhan, Y. (2021). Let knowledge make recommendations for you, *IEEE Access* **9** : 118194–118204.
- Johnson, J. M. and Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance, *Journal of Big Data* .  
**URL:** <https://doi.org/10.1186/s40537-019-0192-5>
- Kalantidis, Y., Sariyildiz, M. B., Pion, N., Weinzaepfel, P. and Larlus, D. (2020). Hard negative mixing for contrastive learning, *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Curran Associates Inc., Red Hook, NY, USA.
- Kardan, A. A. and Ebrahimi, M. (2013). A novel approach to hybrid recommendation systems based on association rules mining for content recommendation in asynchronous discussion groups, *Information Sciences* **219** : 93–110.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0020025512004756>
- Kingma, D. P. and Ba, J. (2014). Adam : A method for stochastic optimization, *CoRR* **abs/1412.6980**.  
**URL:** <https://api.semanticscholar.org/CorpusID:6628106>
- Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks, *arXiv preprint arXiv:1609.02907* .
- Ko, H., Lee, S., Park, Y. and Choi, A. (2022). A survey of recommendation systems : Recommendation models, techniques, and application fields, *Electronics* **11**(1).  
**URL:** <https://www.mdpi.com/2079-9292/11/1/141>
- Koren, Y. (2008). Factorization meets the neighborhood : A multifaceted collaborative filtering model, *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, Association for Computing Machinery, New York, NY, USA, p. 426–434.  
**URL:** <https://doi.org/10.1145/1401890.1401944>
- Koren, Y., Bell, R. and Volinsky, C. (2009). Matrix factorization techniques for recommender systems, *Computer* **42**(8) : 30–37.

- Kotkov, D., Wang, S. and Veijalainen, J. (2016). A survey of serendipity in recommender systems, *Knowledge-Based Systems* **111** : 180–192.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0950705116302763>
- Kunaver, M. and Požrl, T. (2017). Diversity in recommender systems – a survey, *Knowledge-Based Systems* **123** : 154–162.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0950705117300680>
- Kuznetsov, S. and Kordík, P. (2023). Overcoming the cold-start problem in recommendation systems with ontologies and knowledge graphs, *New Trends in Database and Information Systems*, Springer Nature Switzerland, Cham, pp. 591–603.
- Last.fm website* (n.d.). <http://www.lastfm.com>.
- Lerato, M., Esan, O. A., Ebunoluwa, A.-D., Ngwira, S. and Zuva, T. (2015). A survey of recommender system feedback techniques, comparison and evaluation metrics, *2015 International Conference on Computing, Communication and Security (ICCCS)*, pp. 1–4.
- Li, Q., Han, Z. and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI’18/IAAI’18/EAAI’18, AAAI Press.
- Lika, B., Kolomvatsos, K. and Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems, *Expert Systems with Applications* **41**(4, Part 2) : 2065–2073.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0957417413007240>
- Massa, P. and Bhattacharjee, B. (2004). Using trust in recommender systems : An experimental analysis, in C. Jensen, S. Poslad and T. Dimitrakos (eds), *Trust Management*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 221–235.
- Masuda, N., Porter, M. A. and Lambiotte, R. (2017). Random walks and diffusion on networks, *Physics Reports* **716-717** : 1–58. Random walks and diffusion on networks.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0370157317302946>
- Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013). Efficient estimation of word representations in vector space, *Proceedings of Workshop at ICLR* **2013**.

- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. (2013). Distributed representations of words and phrases and their compositionality, in C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Weinberger (eds), *Advances in Neural Information Processing Systems*, Vol. 26, Curran Associates, Inc.
- Mumuni, A. and Mumuni, F. (2022). Data augmentation : A comprehensive survey of modern approaches, *Array* **16** : 100258.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S2590005622000911>
- Natarajan, S., Vairavasundaram, S., Natarajan, S. and Gandomi, A. H. (2020). Resolving data sparsity and cold start problem in collaborative filtering recommender system using linked open data, *Expert Systems with Applications* **149** : 113248.  
**URL:** <https://www.sciencedirect.com/science/article/pii/S0957417420300737>
- O'Donovan, J. and Smyth, B. (2005). Trust in recommender systems, *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI '05, Association for Computing Machinery, New York, NY, USA, p. 167–174.  
**URL:** <https://doi.org/10.1145/1040830.1040870>
- Oord, A. v. d., Li, Y. and Vinyals, O. (2018). Representation learning with contrastive predictive coding, *arXiv preprint arXiv :1807.03748* .
- Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M. and Yang, Q. (2008). One-class collaborative filtering, *2008 Eighth IEEE International Conference on Data Mining*, pp. 502–511.
- Park, D. H. and Chang, Y. (2019). Adversarial sampling and training for semi-supervised information retrieval, *The World Wide Web Conference*, WWW '19, Association for Computing Machinery, New York, NY, USA, p. 1443–1453.  
**URL:** <https://doi.org/10.1145/3308558.3313416>
- Pedrycz, W. (2018). *Granular computing : analysis and design of intelligent systems*, CRC press.
- Perozzi, B., Al-Rfou, R. and Skiena, S. (2014). Deepwalk : online learning of social representations, *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, Association for Computing Machinery, New York, NY, USA, p. 701–710.  
**URL:** <https://doi.org/10.1145/2623330.2623732>

- Pujahari, A. and Sisodia, D. S. (2020). Model-based collaborative filtering for recommender systems : An empirical survey, *2020 First International Conference on Power, Control and Computing Technologies (ICPC2T)*, pp. 443–447.
- Rao, J., He, H. and Lin, J. (2016). Noise-contrastive estimation for answer selection with deep neural networks, *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM '16*, Association for Computing Machinery, New York, NY, USA, p. 1913–1916.  
**URL:** <https://doi.org/10.1145/2983323.2983872>
- Rendle, S., Freudenthaler, C., Gantner, Z. and Schmidt-Thieme, L. (2009). Bpr : Bayesian personalized ranking from implicit feedback, *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI '09*, AUAI Press, Arlington, Virginia, USA, p. 452–461.
- Rennie, J. D. and Srebro, N. (2005). Loss functions for preference levels : Regression with discrete ordered labels, *Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling*, Vol. 1, AAAI Press, Menlo Park, CA.
- Ricci, F., Rokach, L. and Shapira, B. (2010). *Recommender Systems Handbook*, Vol. 1-35, pp. 1–35.
- Roberts, B. and Kroese, D. (2007). Estimating the number of s-t paths in a graph, *J. Graph Algorithms Appl.* **11** : 195–214.
- Robinson, J., Chuang, C.-Y., Sra, S. and Jegelka, S. (2020). Contrastive learning with hard negative samples, *arXiv preprint arXiv :2010.04592* .  
**URL:** <https://doi.org/10.48550/arXiv.2010.04592>
- Safoury, L. and Salah, A. (2013). Exploiting user demographic attributes for solving cold-start problem in recommender system, *Lecture Notes on Software Engineering* **1** : 303–307.
- Sarwar, B., Karypis, G., Konstan, J. and Riedl, J. (2002). Incremental singular value decomposition algorithms for highly scalable recommender systems, *Fifth International Conference on Computer and Information Science* .
- Sharma, R., Gopalani, D. and Meena, Y. (2017). Collaborative filtering-based recommender system : Approaches and research challenges, *2017 3rd International Conference on Computational Intelligence Communication Technology (CICT)*, pp. 1–6.

- Shrivastava, A., Gupta, A. and Girshick, R. (2016). Training region-based object detectors with online hard example mining, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 761–769.
- Shuxian, L. and Sen, F. (2019). Design and implementation of movie recommendation system based on naive bayes, *Journal of Physics : Conference Series* **1345**(4) : 042042.  
**URL:** <https://dx.doi.org/10.1088/1742-6596/1345/4/042042>
- Sun, Z., Deng, Z.-H., Nie, J.-Y. and Tang, J. (2019). Rotate : Knowledge graph embedding by relational rotation in complex space, *International Conference on Learning Representations*.  
**URL:** <https://openreview.net/forum?id=HkgEQnRqYQ>
- Tahmasebi, F., Meghdadi, M., Ahmadian, S. and Valiallahi, K. (2021). A hybrid recommendation system based on profile expansion technique to alleviate cold start problem, *Multimedia Tools Appl.* **80**(2) : 2339–2354.  
**URL:** <https://doi.org/10.1007/s11042-020-09768-8>
- Takács, G., Pilászy, I., Németh, B. and Tikk, D. (2009). Scalable collaborative filtering approaches for large recommender systems, *J. Mach. Learn. Res.* **10** : 623–656.  
**URL:** <https://api.semanticscholar.org/CorpusID:9238574>
- Tamm, Y.-M., Damdinov, R. and Vasilev, A. (2021). Quality metrics in recommender systems : Do we calculate metrics consistently ?, *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, Association for Computing Machinery, New York, NY, USA, p. 708–713.  
**URL:** <https://doi.org/10.1145/3460231.3478848>
- Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. and Mei, Q. (2015). Line : Large-scale information network embedding, *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, p. 1067–1077.  
**URL:** <https://doi.org/10.1145/2736277.2741093>
- Traag, V. (2015). Faster unfolding of communities : Speeding up the louvain algorithm, *Physical Review E* **92**.

Tran, V.-A., Hennequin, R., Royo-Letelier, J. and Moussallam, M. (2019). Improving collaborative metric learning with efficient negative sampling, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, Association for Computing Machinery, New York, NY, USA, p. 1201–1204.

**URL:** <https://doi.org/10.1145/3331184.3331337>

Volkovs, M., Yu, G. and Poutanen, T. (2017a). Dropoutnet : Addressing cold start in recommender systems, in I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (eds), *Advances in Neural Information Processing Systems*, Vol. 30, Curran Associates, Inc.

Volkovs, M., Yu, G. W. and Poutanen, T. (2017b). Content-based neighbor models for cold start in recommender systems, *Proceedings of the Recommender Systems Challenge 2017*, RecSys Challenge '17, Association for Computing Machinery, New York, NY, USA.

**URL:** <https://doi.org/10.1145/3124791.3124792>

Wang, H., Wang, N. and Yeung, D.-Y. (2015). Collaborative deep learning for recommender systems, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, Association for Computing Machinery, New York, NY, USA, p. 1235–1244.

**URL:** <https://doi.org/10.1145/2783258.2783273>

Wang, J., Yu, L., Zhang, W., Gong, Y., Xu, Y., Wang, B., Zhang, P. and Zhang, D. (2017). Irgan : A minimax game for unifying generative and discriminative information retrieval models, *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, Association for Computing Machinery, New York, NY, USA, p. 515–524.

**URL:** <https://doi.org/10.1145/3077136.3080786>

Wang, X., He, X., Wang, M., Feng, F. and Chua, T.-S. (2019). Neural graph collaborative filtering, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, Association for Computing Machinery, New York, NY, USA, p. 165–174.

**URL:** <https://doi.org/10.1145/3331184.3331267>

- Wang, X. and Wang, Y. (2014). Improving content-based and hybrid music recommendation using deep learning, *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, Association for Computing Machinery, New York, NY, USA, p. 627–636.  
**URL:** <https://doi.org/10.1145/2647868.2654940>
- Wang, Y., Wang, L., Li, Y., He, D., Liu, T. and Chen, W. (2013). A theoretical analysis of normalized discounted cumulative gain (ndcg) ranking measures.
- Wu, C.-Y., Manmatha, R., Smola, A. J. and Krähenbühl, P. (2017). Sampling matters in deep embedding learning, *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2859–2867.
- Wu, J., Wang, X., Feng, F., He, X., Chen, L., Lian, J. and Xie, X. (2021). Self-supervised graph learning for recommendation, *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, Association for Computing Machinery, New York, NY, USA, p. 726–735.  
**URL:** <https://doi.org/10.1145/3404835.3462862>
- Wu, S., Sun, F., Zhang, W., Xie, X. and Cui, B. (2022). Graph neural networks in recommender systems : A survey, *ACM Comput. Surv.* **55**(5).  
**URL:** <https://doi.org/10.1145/3535101>
- Wu, T., Chio, E. K.-I., Cheng, H.-T., Du, Y., Rendle, S., Kuzmin, D., Agarwal, R., Zhang, L., Anderson, J., Singh, S., Chandra, T., Chi, E. H., Li, W., Kumar, A., Ma, X., Soares, A., Jindal, N. and Cao, P. (2020). Zero-shot heterogeneous transfer learning from recommender systems to cold-start search retrieval, *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, Association for Computing Machinery, New York, NY, USA, p. 2821–2828.  
**URL:** <https://doi.org/10.1145/3340531.3412752>
- Wu, X., Gao, C., Zang, L., Han, J., Wang, Z. and Hu, S. (2022). ESIMCSE : Enhanced sample building method for contrastive learning of unsupervised sentence embedding, *Proceedings of the 29th International Conference on Computational Linguistics*, International Committee on Computational Linguistics, Gyeongju, Republic of Korea, pp. 3898–3907.  
**URL:** <https://aclanthology.org/2022.coling-1.342>

- Xiao, H., Huang, M. and Zhu, X. (2016). TransG : A generative model for knowledge graph embedding, in K. Erk and N. A. Smith (eds), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, Association for Computational Linguistics, Berlin, Germany, pp. 2316–2325.  
**URL:** <https://aclanthology.org/P16-1219>
- Xiong, L., Xiong, C., Li, Y., Tang, K.-F., Liu, J., Bennett, P. N., Ahmed, J. and Overwijk, A. (2021). Approximate nearest neighbor negative contrastive learning for dense text retrieval, *International Conference on Learning Representations*.  
**URL:** <https://openreview.net/forum?id=zeFrfgYzln>
- Xu, L., Tian, Z., Zhang, G., Wang, L., Zhang, J., Zheng, B., Li, Y., Hou, Y., Pan, X., Chen, Y., Zhao, W. X., Chen, X. and Wen, J.-R. (2022). Recent advances in recbole : Extensions with more practical considerations.
- Yang, Z., Ding, M., Huang, T., Cen, Y., Song, J., Xu, B., Dong, Y. and Tang, J. (2024). Does negative sampling matter? a review with insights into its theory and applications, *IEEE Transactions on Pattern Analysis and Machine Intelligence* .  
**URL:** <https://doi.org/10.48550/arXiv.2402.17238>
- Yang, Z., Ding, M., Zhou, C., Yang, H., Zhou, J. and Tang, J. (2020). Understanding negative sampling in graph representation learning, *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, Association for Computing Machinery, New York, NY, USA, p. 1666–1676.  
**URL:** <https://doi.org/10.1145/3394486.3403218>
- Yang, Z., Ding, M., Zou, X., Tang, J., Xu, B., Zhou, C. and Yang, H. (2023). Region or global? a principle for negative sampling in graph-based recommendation, *IEEE Transactions on Knowledge and Data Engineering* **35**(6) : 6264–6277.
- Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L. and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, Association for Computing Machinery, New York, NY, USA, p. 974–983.  
**URL:** <https://doi.org/10.1145/3219819.3219890>
- Zhang, H., Cisse, M., Dauphin, Y. N. and Lopez-Paz, D. (2018). mixup : Beyond

empirical risk minimization, *International Conference on Learning Representations*.

**URL:** <https://openreview.net/forum?id=r1Ddp1-Rb>

Zhang, W., Chen, T., Wang, J. and Yu, Y. (2013). Optimizing top-n collaborative filtering via dynamic negative item sampling, *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, Association for Computing Machinery, New York, NY, USA, p. 785–788.

**URL:** <https://doi.org/10.1145/2484028.2484126>

Zhang, X., Kuang, Z., Zhang, Z., Huang, F. and Tan, X. (2023). Cold & warm net : Addressing cold-start users in recommender systems, *Database Systems for Advanced Applications*, Springer Nature Switzerland, Cham, pp. 532–543.

Zhou, T., Jiang, L.-L., Su, R.-Q. and Zhang, Y.-C. (2008). Effect of initial configuration on network-based recommendation, *Europhysics Letters* **81**(5) : 58004.

**URL:** <https://dx.doi.org/10.1209/0295-5075/81/58004>

Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R. and Zhang, Y.-C. (2010). Solving the apparent diversity-accuracy dilemma of recommender systems, *Proceedings of the National Academy of Sciences* **107**(10) : 4511–4515.

**URL:** <https://www.pnas.org/doi/abs/10.1073/pnas.1000488107>

Zhu, Z., Kim, J., Nguyen, T., Fenton, A. and Caverlee, J. (2021). Fairness among new items in cold start recommender systems, *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, Association for Computing Machinery, New York, NY, USA, p. 767–776.

**URL:** <https://doi.org/10.1145/3404835.3462948>

## BIOGRAPHICAL SKETCH

Elif Ece Erdem obtained her bachelor's degree in Computer Engineering from Galatasaray University, Faculty of Engineering and Technology, Department of Computer Engineering in Istanbul, Turkey in 2021. She is currently pursuing her master's degree in Computer Engineering at Galatasaray University Institute of Science and Technology. Professionally, Ece works as a research assistant at Galatasaray University and actively contributes to various research studies. Her areas of expertise and interests include data science, data mining, recommender systems, network science, and graph neural networks.

## PUBLICATIONS

- Erdem, E. E., & Orman, G. K. (2023). On the role of user segmentation in recommender systems' performances. *Procedia Computer Science*, 225, 2333-2342.