# Deep Graph Embeddings in Recommender Systems

by

Soon Chee Loong

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

# Abstract

Deep Graph Embeddings in Recommender Systems

Soon Chee Loong
Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2019

Recommender Systems are intelligent machine learning systems that help customers discover a ranked set of personalized products from a dynamic pool of diverse choices. We can formally view recommender systems objective as ranking edges in a bipartite graph consisting of user and item nodes. Deep Graph embeddings have shown great success in a variety of applications. In this thesis, we aim to explore the application of deep graph embeddings for recommendation. We first understand the embeddings from numerous recommendation perspectives. Next, we compare graph embeddings against existing seminal recommendation algorithms on standard recommendation tasks. Then, we design a new evaluation metric that we argue is important for the e-commerce setting. Finally, we extend graph embeddings to improve performance on the recommendation tasks. To conclude, our experiments support using deep graph embeddings for recommendation tasks and indicate that it could be useful to bias recommender models towards long-tail items.

To the reader, who may exist in distant space and time.
To my parents, who never graduated from university but worked hard so that I could.
To my sisters, who would be disappointed if I did not include them here.

# Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Scott Patrick Sanner for all his support, guidance and encouragement throughout my master's research. Scott was always available to meet and consistently provided invaluable feedback towards my research.

I would also like to thank the members of the defense committee, Professor Scott Sanner, Professor Chi-Guhn Lee and Professor Timothy Chan for taking their time to read, and provide insightful research questions about my thesis during my thesis defense as well as providing constructive feedback for my thesis. I would like to thank Professor Christopher Beck for introducing me to the wonderful world of research in 2014, and Professor Jimmy Ba for mentoring and inspiring me to pursue a career in Deep Learning in 2015.

I would like to thank my amazing labmates of the Data-Driven Decision-Making Laboratory, Ga Wu, Dusan, Shagun, Paul, Kasra, Thiago, Brent, Alex, Kai for enjoyable chats and informative lab meetings.

Finally, I would like to thank my former students from MIE250, CSC207, CSC263, CSC258, CSC418 who show up with smiles to tutorials, labs, and office hours during my memorable time here. I would especially like to thank my students from MIE250, who voluntarily voted for me as their favourite Teaching Assistant.

# Contents

# Symbols

| | |
|---|---|
| $R : \|U\| \times \|I\|$ | sparse *implicit* feedback matrix |
| $U$ | set of all users in $R$ |
| $I$ | set of all items in $R$ |
| $r_{u,:}$ | set of observed items interactions for user, $u$ |
| $r_{:,i}$ | set of observed users who interacted with item, $i$ |
| $S^I : \|I\| \times \|I\|$ | item similarity matrix |
| $S^I_{i,j}$ | pairwise similarity score between item $i$ and item $j$ |
| $W^U$ | weight for all users |
| $W^I$ | weight for all items |
| $d(I_i, I_j)$ | distance between items $i$ and $j$ |
| $G = (V, E)$ | binary matrix of unweighted undirected bipartite graph |
| $V : V = V_U \cup V_I$ | set of all vertices |
| $E : \|E\| = \|R\|$ | set of all observed interactions |
| $V_U$ | set of all user vertices |
| $V_I$ | set of all item vertices |
| $A$ | adjacency matrix of $G$ |
| $D$ | degree matrix of $G$ |
| $L$ | Combinatorial Graph Laplacian of $G$ |
| $R^{train}$ | sparse implicit feedback training matrix of $R$ |
| $R^{test}$ | sparse implicit feedback testing matrix of $R$ |
| $\hat{r}_{u,:K} : \|\hat{r}_{u,:K}\| = K$ | set of top-K predicted items for user $u$ |
| $S \in \mathbb{R}^{\|I\|}$ | evaluation score vector |
| $s_i$ | evaluation score of an item $i$ |
| $r^{test}_{u,:K}$ | set of observed items in the test set for user $u$ with the $K$ largest $s_i$ |
| $\hat{r}^{test}_{u,:}$ | set of observed items in the test set for user $u$ descendingly sorted by $s_i$ |
| $tp$ | true positives |
| $tn$ | true negatives |
| $fp$ | false positives |
| $fn$ | false negatives |
| $I^{KNN}_i$ | set of $K$ closest items to item $i$ |
| $C$ | set of all possible class labels |

| | |
|---|---|
| $C_i$ | set of all class labels that item $i$ belongs to |
| $I_c$ | set of items that contains class label $c$ |
| $\odot$ | Hadamard Product |
| $\cdot$ | Dot Product |
| Pop | Popularity |
| KNN | K-Nearest Neighbour [48] |
| PMF | Probabilistic Matrix Factorization [57] |
| WRMF | Weighted Regularized Matrix Factorization [35] |
| PureSVD | Pure Singular Value Decomposition [18] |
| BPR | Bayesian Personalized Ranking [67] |
| DeepRec | Deep AutoEncoder [40, 70] |
| SpectralCF | Spectral Collaborative Filtering [85] |
| GC-MC | Graph Convolutional Matrix Completion [9] |
| DeepWalk | DeepWalk [64] |
| CNN | Convolutional Neural Networks |
| SGNS | Skip-Gram Negative Sampling |

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Recommender Systems for the World

Companies targeting end-users such as Amazon, Netflix and Google have an extremely large number of products. Search engines alone are limited in their ability to navigate through these products as they can only help users query products that they are already aware of. Recommender Systems resolve this limitation by helping users discover items that they may like but are unaware of.

Companies that offer recommender systems for enterprises have also emerged. Amazon Personalize offers real-time personalization and recommendations as a service for companies with no technical experience in recommender systems. Google Ads helps marketers advertise their products to potentially interested customer.

Product recommendations result in larger sales. Netflix estimates that its recommendation is worth about $1 billion [2]. Amazon's recommendations account for 35% of its sales [1]. Google's news recommender generates 38% more clicks [19]. Youtube's video recommender system accounts for 60% of user clicks [20].

There are recommender systems built specifically for various use cases such as books, tags, images, songs, movies, news, travel, social friends, ads and products in e-commerce sites. Each use case contains product-specific characteristics. To illustrate, song recommender systems contains a unique characteristic whereby the same user can listen to the same song more than once whereas product recommendations users do not usually purchase the same product again. News recommender systems need to account for daily changes in the popularity of news to recommend. Travel recommender systems are constrained by location and commute time of travellers. Social recommenders contain nodes representing both users and items.

## 1.2 Graph Perspective of Recommender Systems

Recommender Systems can be viewed as a bipartite graph of user and item nodes. The weighted edges can be viewed as a similarity score between user and item nodes. The graph is bipartite as recommender systems are mainly interested in the connection between user-item pairs. In other words, they only recommend items to users or user to items (e.g. ads).

Figure 1.1 shows a recommender system with a 5 user and 5 items viewed as a bipartite graph. The

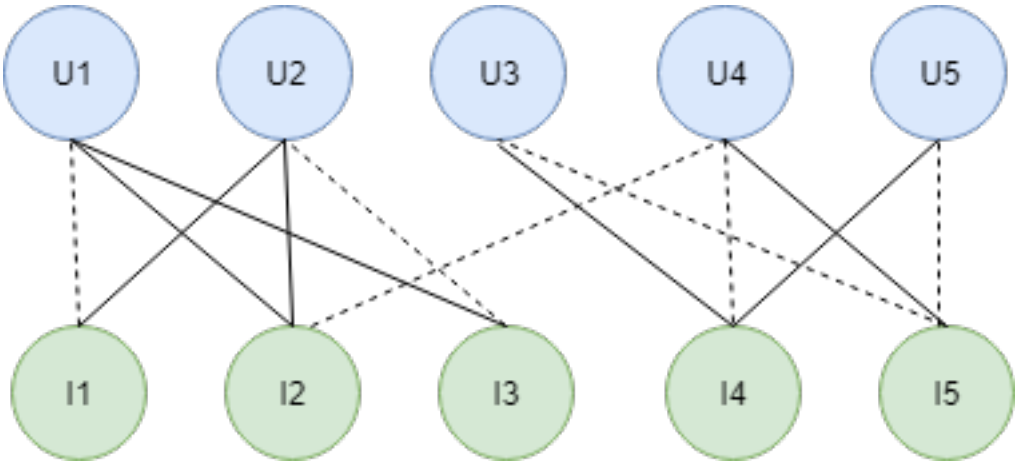



Figure 1.1: Rating matrix as a Bipartite Graph

solid edges indicate a historical interaction such as purchase between the user and item. The dash edges indicate possible future interactions between users and items that need to be predicted. The graph view of recommendation is to rank these edges such that the dash edges result in a higher score. Then, it recommends the highest-scoring non-solid edges for each user node. This view of recommendation allows practitioners to implement graph-based algorithms for the recommendation tasks. Graph-based view also counters the sparsity problem as it can be viewed as a generalization of the rating matrix, whereby a rating matrix is only a first-order connection view of the graph. Sparse first-order connections can be made dense by diving into higher-order connectivities in the graph.

Deep embeddings have shown success in applications such as Computer Vision [38], [32], [47], [6], [65], Natural Language Processing [56], [42], [59], and Speech Processing [33]. Its extension for graph structured data, Deep Graph embeddings has also shown success in graph structured data such as drug discovery [22], [25], semantic segmentation [80], object annotation [49], robot design [77], medical diagnosis [61] and citation networks [29]. This thesis focuses on Deep Graph Embeddings for recommendation as they offer promising new models that have not yet been well studied in the recommendation space. We cover Deep Graph Embeddings in more detail in Chapter 4.

## 1.3 Contributions

The goal of this thesis is to closely study the properties of deep graph embeddings that are not visible from standard end-to-end ranking metrics, compare their performance against state-of-the-art algorithms, and extend them to be more personalized for recommender systems.

### 1.3.1 Research Questions

We aim to answer these research questions:

- **Question 1: What are the properties of graph embeddings?**

  We want to investigate if the embeddings clustered by its recommender system are indeed similar. We can analyze the similarity of embeddings in a supervised manner using labeled item side information. However, we will first need to analyze the correlation between the item embeddings

and their side information to know if it is correct to use them to analyze the similarity of embeddings within a cluster.

Next, we want to examine if it is reasonable to perform mathematical operations such as averaging in the embedding space as it is commonly performed during recommendation [69, 46]. These mathematical operations assume that the embeddings respect our notion of distance. If they do respect our notion of distance, we can further analyze these embeddings using mathematical tools in an unsupervised manner (without using any item side information). Hence, we will first need to analyze how well do the embeddings respect our notion of distance.

- **Question 2: How do graph embeddings compare to state-of-the-art algorithms?**

  We want to compare the performance of graph embeddings against seminal recommender systems under widely used ranking metrics. Additionally, we want to compare their robustness under varying data sparsity.

- **Question 3: How to extend graph embeddings for personalized recommendations?**

  Popularity bias limits personalization in recommender systems. We want to first study the popularity bias of graph embeddings. Then, we want to analyze if it is useful to bias graph embeddings away from the popular items. Then, we want to discover methods to extend graph embeddings to be more user personalized.

### 1.3.2 Contributions

Our contributions to help answer our research questions are below:

**Question 1: What are the properties of graph embeddings?**

- Propose three supervised metrics that use labeled item side information to analyze item embeddings.
- Propose three unsupervised metrics that use mathematical theory to analyze item embeddings.

**Question 2: How do graph embeddings compare to state-of-the-art algorithms?**

- Evaluate the effectiveness of Deep Graph Embeddings on recommender datasets compared to state-of-the-art recommender models.
- Extensive experiments on existing recommender systems on different datasets with varying sparsity.

**Question 3: How to extend graph embeddings for personalized recommendations?**

- Analyze the intensity of embedding bias towards popular items using Recommendation Popularity.
- Propose a new evaluation for evaluating recommender systems on their ability to recommend items in the long tail.
- Propose a new model that extends DeepWalk to prefer items in the long tail.

## 1.4 Outline

This thesis is organized into several chapters.

We cover introductory materials for graph embeddings and recommender systems in chapters 2, 3 and 4. Readers who are already familiar with recommender systems can skip to Chapter 5 onwards. We also cover additional chapter-specific background materials in certain sections of chapters 5, 6 and 7. These background sections are noted in the second paragraph of each of these chapters and can be skipped for those who are already familiar with them. In Chapter 2, we describe the unique characteristics of recommender datasets, analyze the recommender datasets that we use throughout this thesis and emphasize the challenges recommender systems face that are embodied in these datasets. In Chapter 3, we introduce the recommendation problem, approaches to evaluate them, and existing state-of-the-art recommender systems. In Chapter 4, we provide background on Deep Graph Embeddings for recommender systems.

In Chapter 5, we contribute by understanding and analyzing the item embeddings for graph recommenders. The item similarity matrix formed using item embeddings can be used to analyze these embeddings from numerous recommendation perspective such as personalization, interpretability, retrieval, debugging, and scalability. We explain how it is important for the item embeddings to respect metric spaces to make sense in these perspectives. Then, we propose new metrics for analyzing item embeddings using metric spaces and analyze graph item embeddings using these perspectives.

In Chapter 6, we contribute by comparing the effectiveness of deep graph recommenders against existing seminal recommender algorithms under existing well-known ranking metrics. We also compare their robustness towards sparsity, and performances across different values of $K$ as they trade-off between precision and recall.

In Chapter 7, we contribute by extending graph embeddings for recommender systems. We propose an extension to bias training of graph embeddings towards the long tail items. We also propose an evaluation metric for evaluating items on the long tail and evaluate the effectiveness of deep graph recommenders on recommending long-tail items. We also analyze and compare the recommendation popularity of different models to analyze how biased the embeddings are towards popular items. Finally, we show the connections between recommendation popularity and their effectiveness towards recommending long-tail items.

We conclude in Chapter 8, whereby we summarize the main findings of this thesis and highlight avenues for future work.

# Chapter 2

# Recommendation Datasets and Challenges

This background chapter covers properties of recommender datasets and the characteristics of a three publicly available recommender datasets used in this thesis. Readers who are already familiar with recommender datasets can skip to Chapter 5 onwards.

## 2.1 Recommendation Challenges

Recommender systems contains the following challenges embodied in its dataset:

- **Scalability:** Typical recommender system contains a huge number of users and items.
- **Sparsity:** It is cumbersome for any user to interact with every possible available item.
- **Implicit, Binarized, One-Class (positive only feedback):**

  Most datasets only contain positive feedbacks from users. We do not know item a user does not like as a user does not interact with them. We are unaware if the lack of interaction is due to a user not liking an item, or because the user is simply unaware of its existence.

- **Long Tail, Imbalanced:**

  The collected interactions are biased towards a niche set of popular items.

- **Cold Start:**

  No information on new users or items that have not interacted with the recommender system.

## 2.2 Real World Recommendation Datasets

Below, we describe the three real world recommendation datasets used in this thesis.

### 2.2.1 MovieLens

MovieLens is a free movie recommendation service implemented by GroupLens for research purposes. Each user interacts the online recommender system, MovieLens for possible movie recommendations. GroupLens has compiled movie recommendation datasets over different durations of its 17-year service [31]. Each interaction describes a user preference for a particular movie in (userId, movieId, rating, timestamp). Each movie also has additional side information indicating the genre of the movie. Due to computational constraints, we evaluated on only the ml-100k and ml-1m datasets.

### 2.2.2 Amazon Video Games

Amazon is an e-commerce company that capitalizes on product recommendations for users. [54] has collected the recommendations given by Amazon from each particular product pages. Amazon has categorized each product. Due to limited computational resources, we focus only on the Video Games category collected by [54]. Each interaction describes a user's preference for a particular product in (userId, gameId, rating, timestamp). Each game also has additional side information indicating the subcategory within video games that it belongs to.

### 2.2.3 BookCrossing

BookCrossing is a book recommendation engine by Humankind Systems. With permission, [86] has crawled BookCrossing to collect book recommendation data for academic purposes. Each interaction describes a user's rating for a particular book in (userId, bookId, rating). Each book contains additional side information indicating the year it was published.

## 2.3 Dataset Analysis

In this section, we provide a summary statistic of the datasets used throughout this thesis. For computational purposes, we first filter each dataset. Then, the provided summary statistics are based on the statistics of the dataset after filtering.

### 2.3.1 Dataset Filtering

For computational purposes, we filter each dataset by the following steps.

1. Binarize the ratings to observed and unobserved to represent implicit feedback.
2. Maintain only the entries of the 8000 most popular users.
3. Maintain only the entries of the 5000 most popular items.
4. Remove cold users and cold items with no ratings.
5. Split datasets to 80% train and 20% test for each user, based on user interaction timestamps, or randomly if no timestamps are provided.

We do not tackle the cold-start problem in this thesis. Hence, we remove them for computational efficiency as they neither affect our training nor evaluation.

### 2.3.2 Sparsity

Table 2.1 shows the scale and sparsity of the datasets. $|U|$ is the number of warm users, $|I|$ is the number of warm items, $|R|$ is the number of implicit feedback, $density = \frac{|R|}{|U|*|I|}$ and $sparsity = 1.0 - density$. It appears that Amazon Video Games is the most sparse dataset whereas Movielens-100k is the least sparse dataset.

Table 2.1: Scale and Sparsity of Recommender Datasets

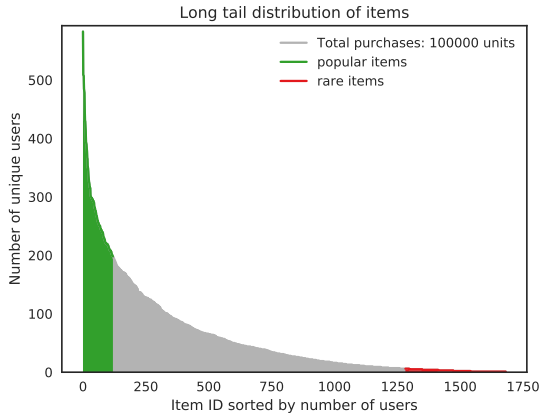
| Dataset | $\|U\|$ | $\|I\|$ | $\|R\|$ | $\frac{\|R\|}{\|U\|*\|I\|}$ |
|---|---|---|---|---|
| MovieLens-100k | 943 | 1682 | 100000 | $6.30 \times 10^{-2}$ |
| MovieLens-1m | 6040 | 3706 | 1000209 | $4.46 \times 10^{-3}$ |
| BookCrossing | 7721 | 5000 | 253967 | $6.58 \times 10^{-3}$ |
| Amazon Video Games | 7926 | 5000 | 107359 | $2.71 \times 10^{-3}$ |

Sparsity is a challenge as it requires recommender systems to learn to generalize across all users. As each user only interacts with a small subset items, the recommender system also needs to learn to be personalized for each user.

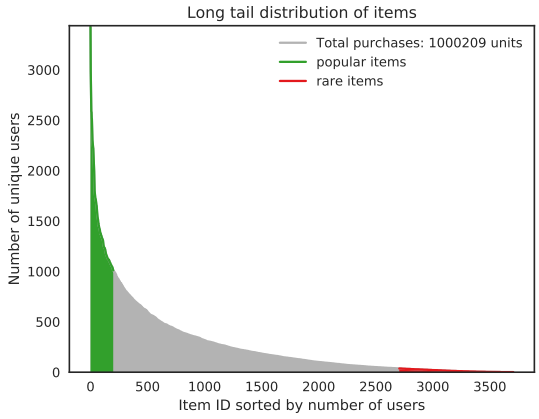
### 2.3.3 Distribution of Interactions

Figure 2.1 shows the distribution of the ratings for all datasets. Each recommendation datasets is shown to exhibit a long tail property.

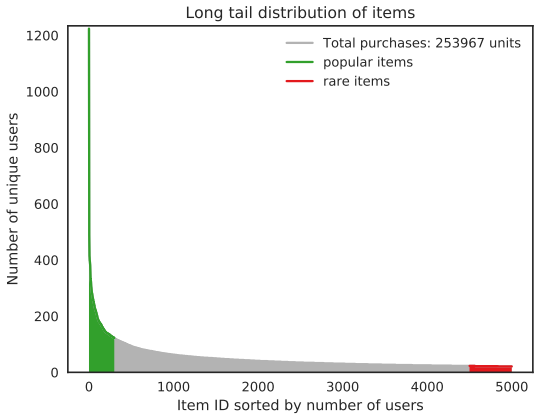
Long-tail introduces a popularity bias. This makes it challenging for recommender systems to learn to be personalized for each user.



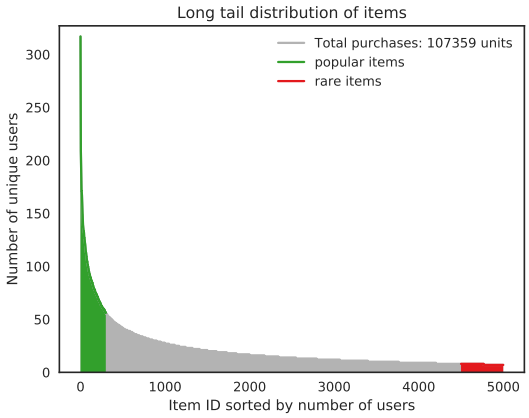

(a) MovieLens-100k Long Tail Distribution




(b) MovieLens-1m Long Tail Distribution




(c) BookCrossing Long Tail Distribution




(d) Amazon Video Games Long Tail Distribution

Figure 2.1: Distribution of Inversely Sorted Popularity of Items. The green region corresponds to the top 5 percentile whereas the red region corresponds to the bottom 10 percentile.

### 2.3.4 Category Labels

Table 2.2 summarizes the categorical labels available for each dataset and the number of items available under each category. These categories are used to analyze the embeddings in Chapter 5. The categories are reversely sorted based on their counts.

Table 2.2: Categories

| $c$ | $\lvert I_c \rvert$ |
|---|---|
| Drama | 725 |
| Comedy | 505 |
| Action | 251 |
| Thriller | 251 |
| Romance | 247 |
| Adventure | 135 |
| Children's | 122 |
| Crime | 109 |
| Sci-Fi | 101 |
| Horror | 92 |
| War | 71 |
| Mystery | 61 |
| Musical | 56 |
| Documentary | 50 |
| Animation | 42 |
| Western | 27 |
| Film-Noir | 24 |
| Fantasy | 22 |

Table 2.3: MovieLens-100k Genres

| $c$ | $\lvert I_c \rvert$ |
|---|---|
| Drama | 1493 |
| Comedy | 1163 |
| Action | 495 |
| Thriller | 485 |
| Romance | 459 |
| Horror | 339 |
| Adventure | 281 |
| Sci-Fi | 274 |
| Children's | 250 |
| Crime | 201 |
| War | 141 |
| Musical | 113 |
| Documentary | 110 |
| Animation | 105 |
| Mystery | 104 |
| Fantasy | 68 |
| Western | 67 |
| Film-Noir | 44 |

Table 2.4: MovieLens-1m Genres

| $c$ | $\lvert I_c \rvert$ |
|---|---|
| 2002 | 528 |
| 2001 | 452 |
| 2003 | 439 |
| 1999 | 364 |
| 2000 | 356 |
| 1996 | 302 |
| 1995 | 297 |
| 1998 | 296 |
| 1997 | 291 |
| 1994 | 243 |
| 1990 | 179 |
| 1993 | 178 |
| 1991 | 167 |
| 1992 | 156 |
| 2004 | 142 |
| 1989 | 115 |
| 1987 | 73 |
| 1986 | 70 |
| 1988 | 70 |
| 1984 | 52 |
| 1983 | 47 |
| 1985 | 47 |

Table 2.5: BookCrossing Publication Year

| $c$ | $\lvert I_c \rvert$ |
|---|---|
| More Systems | 1350 |
| PC | 1109 |
| Xbox 360 | 784 |
| PlayStation 3 | 663 |
| Wii | 352 |
| Nintendo DS | 265 |
| Sony PSP | 120 |
| Nintendo 3DS | 106 |
| PlayStation Vita | 88 |
| Wii U | 56 |
| Xbox One | 46 |
| PlayStation 4 | 38 |

Table 2.6: Amazon Video Games Systems

# Chapter 3

# The Recommendation Problem

In the previous chapter, we covered 3 publicly available recommender datasets that we used throughout this thesis. In this chapter, we cover background on mathematically formalizing the recommendation problem as well as classical algorithms that views recommender systems as a rating matrix.

We start in Section 3.1 by introducing recommender system and its mathematical objective. Then, in Section 3.2, we review approaches to evaluate Top-K recommender systems. Lastly, in Section 3.3, we review the various seminal classical recommender systems used throughout this thesis.

## 3.1 Mathematical Objective of Recommender Systems

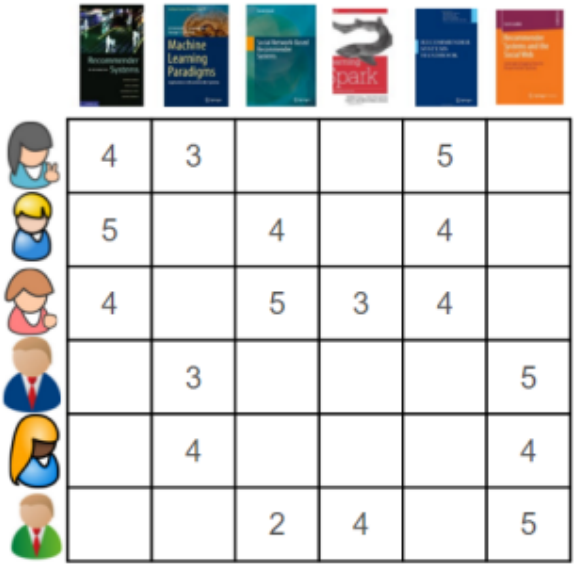



Figure 3.1: Rating Matrix. Taken from [3].

Figure 3.1 shows an example of a rating matrix collected from a book recommender system. Let $R : |U| \times |I|$ be the sparse feedback matrix, whereby $U$ is the set of users in $R$ and $I$ is the set of items in $R$. The entries of this matrix, $r_{u,i}$ represents the rating that user $u$ provides for item $i$.

For recommender systems that only collect implicit feedback, the entries are binary, $r_{u,i} \in 0, 1$ as it can represent either only observed or unobserved interactions. Let $r_{u,:}$ denote the set of observed interactions for user $u$ and $r_{:,i}$ denote the set of users who interacted with item $i$.

The goal of recommendation is to predict the missing entries of $R$. That is, for each user, the recommender system should learn to recommend items that the user may like, but have not yet interacted with. In order to fairly evaluate the performance of the recommender system offline, we divide $R$ into two mutually exclusive sets, $R \subseteq R^{train} \cup R^{test}$, $R^{train} \cup R^{test} \subseteq R$, $R^{train} \cap R^{test} = \emptyset$. The first set, $R^{train}$ is used to train the predictive model, whereas the second set, $R^{test}$ is used to evaluate the trained model.

## 3.2 Evaluation Metrics for Top-K Ranking Recommendation

**Top-$K$ Recommendation** refers to evaluating recommender systems based on their performance on their first $K$ item recommendations. This is useful in cases where users are only displayed $K$ recommendations on each page. Thus, there is no need to evaluate a recommender's performance beyond $K$ recommendations.

Recommender evaluation metrics can be divided into two categories.

- **Error metrics** used in rating prediction are applicable in situations whereby a recommender system displays the average rating on a particular item by past users and tries to predict the rating a new user would give an item.
- **Ranking metrics** are more applicable in situations whereby a recommender system simply displays a ranked list of items that a user may like.

We focus on the latter as rating prediction metrics are not applicable towards implicit recommender datasets.

### 3.2.1 Error Metrics: Rating Prediction

Traditional rating evaluation metrics reduces the distance between predicted ratings and true rating values in the test set. The distance is normally expressed using $L_p$ norm, which satisfies metric spaces properties.

$$\min(||R_{test} - \hat{R}_{test}||_p) \tag{3.1}$$

where $p = 1$ is called Mean Absolute Error whereas $p = 2$ is called Mean Square Error.

However, this metric does not work for implicit recommender datasets that only contains a constant value such as 1 as their observed values. Otherwise, recommender models would only learn the useless trivial solution of predicting a constant value for the ratings of all its recommendations.

Hence, recommender systems use information retrieval metrics to evaluate its top-$N$ recommendations.

### 3.2.2 Set Evaluation Metric

Set evaluation metrics are metrics which evaluate a set of predicted items compared to a set of test items. A set contains no duplicates and there is no notion of order of items in a set. Hence, set evaluation

metric only cares about the presence or absence of elements in a set. This assumption is beneficial for implicit feedback recommender datasets that only contains the presence and absence of user interaction signals.

Referring to Figure 3.2, the top green region is the region of all observed test items that users did interact with whereas the bottom red region is the region of all unobserved test items that users did not interact with. The blue region is the items that the recommender system predicted a user would want to interact with. Hence, we can divide up the items into 4 non-intersection categories.

- **True Positive (tp):** Observed items that are correctly predicted as observed.
- **True Negative (tn):** Unobserved items that are correctly predicted as unobserved
- **False Positive (fp):** Unobserved items that are wrongly predicted as observed.
- **False Negative (fn):** Observed items that are wrongly predicted as unobserved.

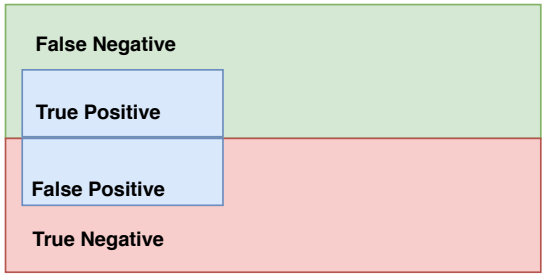



Figure 3.2: Confusion Matrix

**Accuracy:** measures the number of correctly predicted labels.

$$Accuracy = \frac{tp + tn}{tp + fp + tn + fn} \tag{3.2}$$

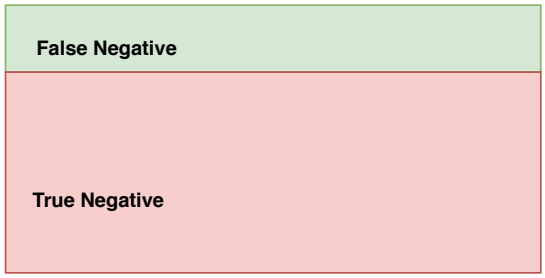



Figure 3.3: Unbalanced Confusion Matrix

However, accuracy is flawed for many machine learning tasks due to the data imbalanced problem, whereby the number of true negatives is significantly greater than the number of true positives. This phenomenon is shown in Figure 3.3, whereby the number of unobserved items is significantly higher than the number of observed items. Therefore, this leads to models that can easily be highly accurate by the trivial solution of predicting that a user does not like any item. Since $tn > fn$, this trivial solution would still have high accuracy. Therefore, metrics which ignores the presence of $tn$ help to counteract this data imbalanced issue.

**Precision:** measures the accuracy within the set of predicted items.

$$Precision = \frac{tp}{tp + fp} \tag{3.3}$$

Hence, it bypasses the data imbalanced problem as it ignores the presence of $tn$. However, **Precision** has its own limitation. A recommender model could now choose the trivial solution of only predicting the top 1 item it is highly confident is going to be observed, leading to a perfect score of precision. Precision misses out on a large number of observed test items available to predict.

**Recall:** measures the number of observed items in the test set that were predicted as observed. Thus, **Recall** overcomes the limitation of precision. However, a trivial solution of predicting everything as positive would lead to a perfect score for **Recall**.

$$Recall = \frac{tp}{tp + fn} \tag{3.4}$$

Hence, a metric that balances between **Precision** and **Recall** is the final solution to evaluate imbalanced sets. Commonly used in binary classification task is the $F_1$**-Measure**, which is simply the harmonic mean between **Precision** and **Recall**.

### Set Evaluation for Recommender Systems

In order to adapt set evaluation metric from the Information Retrieval literature into the recommendation space, we will need to define *hit* and *miss* in recommendation. In other words, we will need to define *tp*, *fp*, *tn* and *fn* for recommender systems.

Let $\hat{r}^{test}_{u,:}$ = be the set of predicted item interactions for user, $u$ ordered descending by their relevance. In other words, $\hat{r}^{test}_{u,:1}$ = is the item with the highest confidence for relevance for user $u$ that is not in $u$'s training set.

Let $r^{test}_{u,:}$ = be the set of observed item interactions for user, $u$ ordered descending by their relevance. For implicit feedback, it is all the observed items first, preceded by the unobserved items that are not in $u$'s training set.

We can think of *hit* and *miss* as throwing darts. The number of darts we throw for each user is determined by $K$, the number of top items we recommend. Hence, the top-$K$ items being thrown for user $u$ is given by $\hat{r}^{test}_{u,:K}$. Hence, these $K$ items are being predicted as positives. Then, every other available observed item in the test set for user $u$, $r^{test}_{u,:} \setminus \hat{r}^{test}_{u,:K}$ are thought of as being predicted as negatives. A hit occurs if that item is in the observed test set, whereby we don't just look at the top-$K$ in the test set as we assume implicit feedback, where all observed items have equal score and importance in the test set (note: This assumption is not valid in a later chapter we discuss called *Profitability*).

Hence,

$$\begin{aligned} hit@K_u &= \hat{r}^{test}_{u,:K} \cap r^{test}_{u,:} \\ miss@K_u &= \hat{r}^{test}_{u,:K} \setminus r^{test}_{u,:} \end{aligned} \tag{3.5}$$

$K$ can be viewed as a threshold for decision making on the number of items to recommmend and treated as positives.

$$
\begin{aligned}
Precision@K_u &= \frac{|hit@K_u|}{K} \\
Precision@K = P@K &= \frac{1}{|U|}\sum_{u=1}^{|U|} Precision@K_u
\end{aligned}
\tag{3.6}
$$

$$
\begin{aligned}
Recall@K_u &= \frac{|hit@K_u|}{|r^{test}_{u,:}|} \\
Recall@K = R@K &= \frac{1}{|U|}\sum_{u=1}^{|U|} Recall@K_u
\end{aligned}
\tag{3.7}
$$

Thus, we can infer from equations 3.6 and 3.7 that increasing $K$ tends to increase **Recall** but decrease **Precision**.

We can instead choose to decide the threshold to be different for each user that is based on test data, whereby we want to make use of all possible observed test items.

***R*-Precision:** $R$-Precision has flexibility in varying the chosen $K$ in **Precision@**$K$ to be different for each test user. This is useful because every user has a different number of observed test items.

Let $R$ is set to be the number of observed test items available for the current user.

$$
\begin{aligned}
R &= |r^{test}_{u,:}| \\
hit@R_u &= \hat{r}^{test}_{u,:R} \cap r^{test}_{u,:} \\
Precision@R_u &= \frac{|hit@R_u|}{R} \\
RPrecision &= \frac{1}{|U|}\sum_{u=1}^{|U|} Precision@R_u
\end{aligned}
\tag{3.8}
$$

From referring to equations 3.6, 3.7 and 3.8, if we vary $K$ similar to $R$-Precision, we can infer that $Precision@R = Recall@R$ since the denominators are the same for each user under both metrics.

$$
RPrecision = Precision@R = Recall@R \tag{3.9}
$$

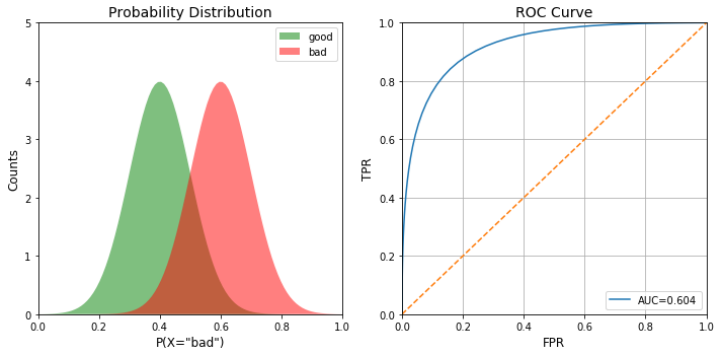



Figure 3.4: Receiver Operating Characteristic Curve

However, set based measure depends on a specific threshold for binary decision making. R-Precision is biased towards the largest possible $K$ for evaluation. To illustrate, Figure 3.4 shows a hypothetical Receiver Operating Characteristic (ROC) curve. The left figure shows the distribution of positive items

in green and negative items in red. The right figure shows the ROC Curve. The x-axis shows the false positive rate and y-axis shows the true positive rate. Rate is useful in normalizing the data imbalanced between positive and negative such that it does not affect the scale of the plot. Initially, the threshold for negative is 0. Hence, all items are being predicted as negative. Initially, both the true positive rate and false positive rate are 0 as it does not predict anything as positive. As you increase the threshold for negative, the number of items predicted as positive increases. Hence, the true-positive rate and false-positive rate can only monotonically increase. The performance is then evaluated based on how much higher the true-positive rate is compared to the false-positive rate at each threshold. Thus, the diagonal dotted line in orange shows a random classifier, which has somewhat equal true-positive and false-positive rates.

This is problematic for set-based measures such as $F_1$**-Measures** as each threshold can result in a different score. In the context of recommendation, lowering the number of top recommended items, $N$ results in a low negative threshold, which increases precision. On the other hand, increasing $N$ increases recall. A specific threshold exists that maximizes $F_1$**-Measure**.

Thus, the limitations of set-based measures are:

- **Threshold dependent:** Set-based evaluations are dependent on a specifically chosen threshold (i.e. choice of $N$), and each recommender model can have a different maximizing threshold.

- **Order independent:** Set-based measures do not account for the order of the recommended items.

### 3.2.3 Ranking Binarized Evaluation Metric

Ranking binarized evaluation metrics are metrics that account for the ranking of items in recommendation. In other words, it accounts for the order of items within its recommendation. It is binarized as each item in the rank is equally weighted. For instance, the presence or absence of an item in the rank is a binary value.

**Area Under ROC Curve (AUC-ROC):** For comparison of different recommender model purposes, it suffices to just compare how well both models perform on average over its range of possible scores based on the entire domain of possible thresholds. Hence, a model with a large area under ROC curve performs better as its true-positive rate increases faster on average compared to its false-positive rate. The drawback of AUC-ROC is that it does not account for the data imbalanced problem in recommender datasets.

**Area Under PR Curve (AUC-PR):** The area under the Precision-Recall curve is more informative when evaluating binary classifiers on imbalanced datasets as it accounts for the dataset imbalanced [21, 68]. The drawback is that it is computationally intensive to plot and calculate the AUC-PR for each recommender model.

**AveragePrecision@$K$:** Average precision can be thought of as an approximation of the AUC-PR as it computes the arithmetic mean of precision scores across different possible thresholds. Notice, that the order within the $K$ recommendations now matters compared to just using **Precision@$K$**.

$$
\begin{aligned}
AveragePrecision@K_u &= \frac{1}{K}\sum_{k=1}^{K} Precision@k_u \\
mAP@K = meanAveragePrecision@K &= \frac{1}{|U|}\sum_{u=1}^{|U|} AveragePrecision@K_u
\end{aligned}
\tag{3.10}
$$

### 3.2.4 Ranking Graded Relevance Evaluation Metric

Ranking Graded Relevance Evaluation Metric accounts for varying scores for each item in the ranked list. For instance, in a rating matrix, an item with a higher rating value should result in a higher evaluation score compared to another item with a lower rating value. Hence, it accounts for the importance of each relevant item while accounting for the order of the items in recommendation.

**Regret:** originate from the bandit literature [52, 11]. It compares the performance of an algorithm from the optimal solution, where regret is associated with the amount of loss that results from not choosing the most optimal solution available at the current time step. It is also more intuitive for online algorithms that are evaluated sequentially at every timestep. Regret for recommender systems is defined as [53, 78]:

$$
\begin{aligned}
regret_{u_t} &= \max_{i_t^*}(r_{u,i_t^*}) - r_{u,i_t} \\
regret_u &= \sum_{t=1}^{T} regret_{u_t} \\
regret &= \frac{1}{|U|}\sum_{u\in U} regret_u
\end{aligned}
\tag{3.11}
$$

where $i_t$ is the selected item at the current timestep and $i_t^*$ is the item with the highest possible rating in the test set for user, $u$ that has not yet been selected at a previous timestep. $T$ is equivalent to $R$ from $R$-**Precision** for each user.

**Normalized Discounted Cumulative Gain (NDCG):** discounts the gain for relevant items that are at the bottom of the ranked list. It is also normalized by the ideal ranked list of recommendation.

$$
\begin{aligned}
dcg_u &= \sum_{t=1}^{T} \frac{r_{u,i_t}}{log_2(t+1)} \\
idcg_u &= \sum_{t=1}^{T} \frac{r_{u,i_t^*}}{log_2(t+1)} \\
ndcg_u &= \frac{dcg_u}{idcg_u} \\
ndcg &= \frac{1}{|U|}\sum_{u\in U} ndcg_u
\end{aligned}
\tag{3.12}
$$

NDCG differs from **Regret** as it takes a more optimistic view of the score. Evaluation could either try to maximize **NDCG** or minimize **Regret**. Also, regret penalizes algorithms that do not pick the best available rated item first as that item's rating would continually be used in regret calculation for every timestep.

**Regret** is used in bandit algorithms applied to recommender systems [84, 46, 36] whereas NDCG is used in the standard top-$N$ recommendation.

Since this thesis focuses on implicit feedback datasets, there is no need to account for graded relevance in our evaluation.

## 3.3 Classical Recommender Systems

This section covers existing seminal baselines for recommender systems used throughout this thesis.

**Popularity:** a non-user-personalized algorithm that recommends the same set of top K most popular items across all users in the training set.

$$\underset{i}{\text{argmax}}(P(hit|item = i)) = \underset{i}{\text{argmax}}(\frac{|r_{:,i}|}{|U|}) = \underset{i}{\text{argmax}}(|r_{:,i}|) \tag{3.13}$$

**K Nearest Neighbour Collaborative Filtering [48]:** is a recommender algorithm that recommenders solely based on the similarity matrix formed using Nearest Neighbours. An item-based nearest neighbour approach is used when the number of items is less than the number of users. Given a user, $u$ and an item $i$. Let $kNN(i)$ be the set of $k$ closest items to item $i$, whereby closeness is defined by an appropriate similarity score that forms the pairwise item-item similarity matrix, $S^I$ (details can be found in 5.1.2). The predicted rating is then calculated as in Equation 3.14.

$$\hat{R}_{u,i} = \frac{\sum_{j \in kNN(i)} R_{u,j} \cdot S^I_{i,j}}{\sum_{j \in kNN(i)} S^I_{i,j}} \tag{3.14}$$

**Deep AutoEncoder [40, 70]:** is an encoder-decoder approach trained by simply re-predicting the users that purchased each item for each item. The encoders and decoders are formed using Fully Connected Neural Networks. Let $W_{enc}$ be the encoder weights, $W_{dec}$ be the decoder weights, and $f_{enc}$ and $f_{dec}$ be their corresponding non-linear activating functions. The input is reconstructed as Equation 3.15. It is trained to minimize a loss function between the reconstructed input $\hat{R}$, and the actual input, $R$.

$$\hat{R} = f_{dec}(W_{dec} \cdot f_{enc}(W_{enc} \cdot R)) \tag{3.15}$$

**Matrix Factorization:** approaches linearly factorizes the rating matrix, $R$ into two lower-dimensional representations $R \approx W^U \cdot (W^I)^T$. $W^U \in R^{|U| \times d}$ can be viewed as the latent representation of users whereas $W^I \in R^{|I| \times d}$ can be viewed as a representation of items. Since each user and item representation are of equal dimension, $d$, we can reconstruct the user-item matrix by a simple dot product $\hat{R} = W^U \cdot (W^I)^T$. Since the matrix is sparse, matrix factorization approaches require that $d < \min(|U|, |I|)$ to not overfit and be able to generalize to learn the unobserved entries in $R$. Below are widely used approaches to computing the latent representations $W^U$ and $W^I$ for recommender systems:

- **Probabilistic Matrix Factorization [57]:** is a seminal recommender algorithm that trains using backpropagation on only the explicit observed entries ($|R_{u,i}| > 0$). It avoids the inefficiency of alternating training between user and item embeddings. However, it does not account for the unobserved entries. It optimizes Equation 3.16 using stochastic gradient descent.

$$\underset{W^U, W^I}{\text{argmin}} \sum_{u \in U, i \in I} (|R_{u,i}| \times ||R_{u,i} - W^U_u \cdot W^I_i||) \tag{3.16}$$

- **Weighted Regularized Matrix Factorization (WRMF) [35]:** is a seminal recommender algorithm that is still competitive [76]. It is a Locally Weighted Learning [16] approach towards recommendation. It is seminal as it is the first to successfully incorporate unobserved entries in its optimization. Yet, it is still scalable as the math derivation reduces to only training on the observed entries. Its optimization places more focus on higher rating levels as shown in Equation 3.17, where $\alpha$ is a hyperparameter.

$$\begin{aligned} c_{u,i} &= 1 + \alpha * r_{u,i} \\ W^U, W^I &= \underset{W^U, W^I}{\operatorname{argmin}} \sum_{u,i} c_{u,i}(r_{u,i} - (W_u^U)^T \cdot W_i^I)^2 \end{aligned} \tag{3.17}$$

  WRMF accounts for the unobserved ($r_{u,i} = 0$) entries as they contribute a constant weight of $c_{u,i} = 1$ in the optimization. However, the closed-form solution beautifully cancels this constant out and hence is scalable as it only trains on the observed entries. More details can be found in [35].

- **Pure Singular Value Decomposition [18]:** is a linear dimensionality reduction algorithm that simply factorizes the recommender matrix using just purely Singular Value Decomposition. It is a seminal algorithm which highlighted that simply using SVD performs well on ranking metrics and does not need to be specifically optimized for the rating prediction tasks.

$$SVD(R) = W^U \cdot \Sigma \cdot (W^I)^T \tag{3.18}$$

  where $W^U \in \Re^{|U| \times d}$ are the user embeddings and $W^I \in \Re^{|I| \times d}$ are the item embeddings, where we keep the vectors that corresponds to the largest $d$ singular values in $\Sigma$.

- **Bayesian Personalized Ranking [67]:** is also a seminal recommender algorithm that was the first to directly optimize for ranking in its loss function. It probabilistically reasons that observed items should always rank higher than unobserved items for each user as shown in Figure 3.5.

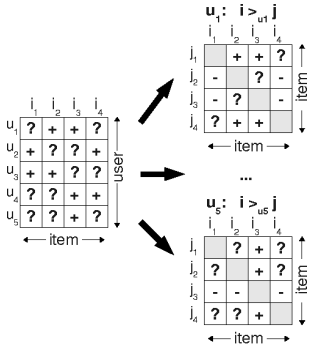



Figure 3.5: Rating matrix is split into ordered relations. Taken from [67].

# Chapter 4

# Deep Graph Recommender Algorithms

In the previous chapter, we covered the basics of viewing recommender system as a rating matrix, the approaches to evaluate recommender systems, and classical seminal algorithms that views recommender systems as a rating matrix used throughout this thesis. This chapter covers deep graph embeddings that views recommender systems as a bipartite graph. Readers who are already familiar with deep graph embeddings can skip to Chapter 5 onwards.

We start in Section 4.1 by reviewing the two successes of Deep Learning approaches, mainly in Image and Natural Language domains. Then, in Section 4.2, we review approaches to generalize the deep learning approaches in both domains to graph-structured data for recommender systems.

## 4.1 Success of Deep Learning

Deep Learning has recently achieved state-of-the-art performance on image [38] and natural language [56] domains thanks to the advent of GPU and availability of large datasets that enable large-scale training. We focus on analyzing the effectiveness of Deep Learning approaches when applied to recommender datasets.

### 4.1.1 Image Domains

Convolutional Neural Networks (CNN) has achieved success in various image tasks such as Image Classification [32], Object Detection [47], Image Segmentation [6] and Image Generation [65].

Convolutional Neural Networks [44] improves on Fully Connected Neural Networks (FCNN) in the image domain due to its ability to share weights, learn interpretable feature detectors [72], and is translation-invariant towards positions of objects in an image. Deeper layers also imply composition in building image features.

Convolution operator enables these abilities. Convolution applied to images can be thought of as sliding a set of feature masks through different regions of an image, which activates when a significant feature is found in an image. For instance, there could be a feature mask that activates whenever it detects an eye in an image. The learned feature detectors are then successfully applied to classification,

detection, segmentation, or generation for images.

CNN has successfully learned useful nonlinear embeddings that generalize well to higher-level image tasks. This suggests that it may be promising to use CNN to learn embeddings for the recommendation task.

### 4.1.2 Natural Language Domains

In natural language domains, Skip-Gram Negative Sampling (SGNS) has achieved success in machine translation [55], named entity recognition [42], text summarization [59] and medical diagnosis [14].

SGNS is a deep learning approach that learns useful word embeddings from a corpus of sentences. SGNS inherently loops through each context, and train each word in the context to have similar embeddings to every other word in the context [56]. The learned word vectors can then be used as embeddings for higher-level language tasks.

SGNS has successfully learned useful embeddings for higher-level language tasks. This suggests that it may be promising to use SGNS to learn embeddings for the recommendation task.

## 4.2 Deep Graph Embedding

In the previous section, we covered successes of CNN in the image domain and successes of SGNS in natural language domain. However, CNN is trained on labelled images, whereas SGNS is trained on a corpus of words. This is not directly applicable to the recommender domain, where we only have access to a rating matrix, $R$. Existing CNN and SGNS only works on Euclidean domain (grids).

In this section, we derive 3 different deep graph embedding models from the successes of CNN in image domain, and SGNS in the natural language domain.

- **Spectral Graph Convolution** performs exact convolution in the frequency domain, but it is computationally expensive.
- **Spatial Graph Convolution** performs computationally efficient convolution in the spatial domain, but it only an approximation of full convolutions.
- **DeepWalk** is a SGNS based network embedding approach.

In Section 4.2.1, we review approaches to generalize convolution in the image domain to both Spectral Convolution and Spatial Convolution in graph domains. In Section 4.2.2, we review approaches to generalize SGNS in natural language domain to graph domains. In Section 4.2.3, we discuss possible extensions for the Deep Graph Architecture in the context of recommender systems.

### 4.2.1 Generalizing Convolution to Recommender Systems

The goal is to generalize the use of CNN in the image domain to the recommender domain. This section leads to the derivation of **SpectralCF** and **GCMC**, two of our Deep Graph Embedding models used throughout this thesis.

We first review the definition of Convolution in Euclidean domain and also state its limitations for direct use in general graphs. Then, we use **Convolution Theorem** to perform the equivalent of spatial convolution in the spectral domain. Then, we use **Spectral Graph Theory** to define Spectral

Transform that is needed for transforming between spectral and spatial domains. This leads to the derivation of **Spectral Graph Convolution**.

We then review approaches to address the computational limitation of exact spectral graph convolution. We review Polynomial Parameterization with Chebyshev Expansion that can trade-off between model complexity and computational complexity using a model hyper-parameter, $K$. Then, we show that we can further speed up computations by replacing sequential operations with parallelizable computation. We can do this by limiting $K = 1$ and instead increase the model depth to maintain model complexity. These lead to the derivation of **Spatial Graph Convolution**.

Lastly, we review the model architecture of **SpectralCF**, an instance of Spectral Graph Convolution that is most widely used for recommender systems We also review the model architecture of **GCMC**, an instance of Spatial Graph Convolution that is most widely used for recommender systems.

### Convolution in Euclidean Spatial Domain and limitations for General Graphs

Convolution is defined on the Euclidean spatial domain as

$$output[x, y] = \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} kernel[i, j] * input[x - \left\lfloor \frac{k}{2} \right\rfloor + i, y - \left\lfloor \frac{k}{2} \right\rfloor + j] \tag{4.1}$$

where $k$ is the kernel filter size.

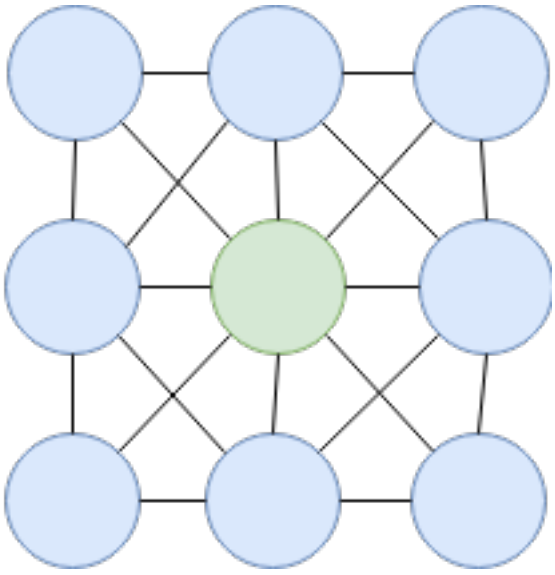


Figure 4.1: 3x3 image as a graph

We can view an image as a graph whereby each pixel are nodes and each pixel are connected by an edge as shown in Figure 4.1. Equation 4.1 is defined for discrete images.

However, this requires a few assumptions that do not hold for general graphs.

- First, it assumes each node (refer to the green node in Figure 4.1) are always connected by 8 edges (which is true for non-corner pixels). However, a general node in a graph can contain an arbitrary number of edges.

- Next, it assumes that there is a specific responsibility of each node when sliding filters to each node in a graph. For instance, there is always a middle node, an upper left node, an upper right node as it assigns a specific share weight for the upper left node, middle node.
- Next, it relies on a notion of ordering between nodes as it slides across the image to the right and to the bottom to form the output, resulting in the receptive field being more focused towards the center pixels than the corner pixels [51]. However, we can't translate on an arbitrary graph as there is no way to specify that one node is in the center of another node.
- Lastly, the stride operation of CNN which downsamples by skipping adjacent nodes does not make sense in graphs as Nyquist Sampling Theorem would not hold for nodes without order. Also, it may not maintain the original structural properties of an arbitrary graph. Besides, we are not guaranteed to ensure every node is accounted for in the receptive field of deeper layers from arbitrary skipping on arbitrary graphs.

These assumptions do not hold for general graphs. Therefore, we need to define convolution in a way that holds for general graphs.

#### Convolution Theorem on the equivalence between Spatial and Spectral Convolution

It is unclear what convolution means in the spatial domain, however, we know what Hadamard product means in the spectral domain. As a consequence of **Convolution theorem**, the convolution between two signals is a point-wise product in their Fourier Transform. This means we can first view the nodes in a graph as a signal and the filter as another signal. Then, their spatial convolution is well defined by simply taking their Hadamard Product of these signals Fourier Transform, and taking an inverse Fourier Transform to return into the spatial domain. In other words, we can use **Graph Signal Processing** to first transform the signals into the spectral domain, take a Hadamard product, and transform back into the spatial domain. This way, we would have performed convolution in the spectral domain.

#### Spectral Graph Theory on Graph Fourier Transform

We need to know how to take the Fourier Transform for Graphs. We first review Combinatorial Laplacian, which is used by Spectral Graph Theory to define Graph Fourier Transform.

The Combinatorial Laplacian is defined as $L = D - A = I - D^{-1} \cdot A$, where $A$ is the graph's adjacency matrix whereas $D$ is the graph's degree matrix. $L$ is a symmetric matrix (since both $D$ and $A$ are symmetric). Hence, it has orthonormal bases and real non-negative eigenvalues. $L = U \cdot \lambda \cdot U^T$ where $U \cdot U^T = I$. **Spectral Graph Theory** [17] has defined a notion of frequency for graphs. The notion of frequency comes from the eigenvalues as larger eigenvalues exhibit larger frequency due to the larger number of zero crossings on the graph. Thus, [71] defines **Graph Fourier Transform** on a graph signal $x$ as $\hat{x} = U^T \cdot x$, where the Inverse Graph Fourier Transform is then $x = U \cdot \hat{x}$.

**Spectral Graph Convolution:** The convolution between two signals can be defined as the Hadamard product in their spectral domain.

$$x * y = U \cdot (U^T \cdot x) \odot (U^T \cdot y)) = U \cdot (\hat{x} \odot \hat{y}) \tag{4.2}$$

The Spectral-domain is spanned by eigenvectors corresponding to different frequency eigenvalues. A

frequency filter on the signals would alter each frequency component (eigenvalue) independently of others. A signal filtered by $f_\theta$ is:

$$x_{filter} = f_\theta(L) \cdot x = f_\theta(U \cdot \lambda \cdot U^T) \cdot x = U \cdot f_\theta(\lambda) \cdot U^T \cdot x = U \cdot f_\theta(\lambda) \cdot \hat{x} \tag{4.3}$$

whereby $f_\theta(\lambda) = diag(\theta)$

The problem with a full frequency filter as shown in Equation 4.3 is that it is not localized in space, and requires too many parameters, one for each node. Its learning complexity is $O(|V|)$. Hence, it is not adaptive to new user and item nodes or scalable to a large number of nodes. Therefore, we follow the lead of others [22] to filter using a **Polynomial Parameterization** approximation instead.

$$f_\theta(\lambda) = \sum_{k=0}^{K-1} \theta_k L^k \tag{4.4}$$

**K-Localized Filters:** The size of parameters grows with our own chosen polynomial order, $K$, with learning complexity $O(K)$. We also can view $K$ as the number of edge traverses connecting one node to another as each power on $L$ connects each node by another edge. Therefore, the model can be said to be K-localized. Taking the $k^{th}$ power of a matrix can be slow, but in this case, it is fast due to taking powers of only a diagonal matrix. Using $U \cdot U^T = I$ results in,

$$f_\theta(\lambda) = \sum_{k=0}^{K-1} \theta_k L^k = \sum_{k=0}^{K-1} U \cdot \theta_k \lambda^k \cdot U^T = U \cdot (\sum_{k=0}^{K-1} \theta_k \lambda^k) \cdot U^T \tag{4.5}$$

The Polynomial approximation greatly reduces the learning complexity of the model from $O(|V|)$ to $O(K)$. However, the time complexity of the model is still high with $O(|V|^2)$. A solution proposed by [22], called **ChebNet** is to use the Chebyshev expansion [30].

$$\begin{aligned} T_0(x) &= 1; \\ T_1(x) &= x; \\ T_t(x) &= 2xT_{t-1}(x) - T_{t-2}(x); \end{aligned} \tag{4.6}$$

Hence, the filter can be recursively parameterized with

$$f_\theta(\lambda) = \sum_{k=0}^{K-1} \theta_k \cdot U \cdot T_k(\tilde{L}) \cdot U^T \tag{4.7}$$

where $\tilde{L} = \frac{2}{\lambda_{\max}} L - I$ for numerical stability. Therefore, its computational complexity is now $O(K|E|)$ instead of $O(|V|^2)$, which is linear in the number of observed edges while maintaining Note that the embeddings do not lose representational power as they are still **K-Localized** since it's still taking $K^{th}$ power of the Laplacian.

To further speed up computation, [37] propose to limit $K = 1$. Relying on Chebyshev approximation results in sequential computation, which isn't parallelizable. Limiting $K = 1$ allows parallelizable computation. However, the embeddings are now only first-order hops away instead of higher-order hops. Therefore, [37] also suggested forming deeper layers of $K = 1$ Graph Convolution to enable higher-order

hops while maintaining the parallelizability of each layer. Further simplifying $\lambda_{\max} = 2$ results in

$$f_\theta * x \approx \theta_0 + \theta_1(L - I) \cdot x = \theta_0 - \theta_1 D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}} \cdot x \tag{4.8}$$

Further simplifying $\theta = \theta_0 = -\theta_1$ results in

$$f_\theta * x \approx \theta(I + D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}}) \cdot x \tag{4.9}$$

However, the eigenvalues are now in the range $[0, 2]$, which can lead to numerical instability. Therefore, [37] also propose a renormalization trick to set

$$(I + D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}}) = (D^{-\frac{1}{2}} \cdot (A + I) \cdot D^{-\frac{1}{2}}) \tag{4.10}$$

Hence, the final filter at each layer is then

$$Z = (D^{-\frac{1}{2}} \cdot (A + I) \cdot D^{-\frac{1}{2}}) \cdot X \cdot \theta \tag{4.11}$$

**Spatial Graph Convolution:** Convolution in the spatial domain can then be viewed as a first order approximation as convolution in the spectral domain.

$$\begin{aligned} \hat{A} &= D^{-\frac{1}{2}} \cdot (A + I) \cdot D^{-\frac{1}{2}} \\ Z &= \hat{A} \cdot X_l \cdot W \\ X_{l+1} &= \sigma(Z) \end{aligned} \tag{4.12}$$

Equation 4.12 above demonstrates that each node's next layer embedding is now a weighted linear combination of its neighbours' previous layer state embedding and passing through a non-linearity, $\sigma$. The normalization trick can be thought of as self-edge for each node to itself. Therefore, Graph Convolution in the spatial (vertex) domain is simply centering each convolution on each node, and taking a localized weighted combination of each node's first order neighbour at each layer. Deeper layers imply higher-order convolution. Although Spatial Graph Convolution can be viewed as an approximation to Spectral Graph Convolution, it should not be strictly as being inferior to spectral graph convolution as it is less prone to overfitting compared to spectral graph convolution.

Below, we discuss the existing algorithms that apply **Graph Convolutional Networks** in the recommender domain, mainly **SpectralCF** (spectral-based) and its alternative, **GCMC** (spatial-based).

#### Spectral Collaborative Filtering (SpectralCF) [85]

**SpectralCF** is the most popular instance of Spectral Graph Convolution for recommender systems. Thus, it is a full frequency filter but has scalability drawbacks. Following Equation 4.3 and using Polynomial approximation as shown in Equation 4.5 with $K = 2$ results in Equation 4.13.

$$
\begin{aligned}
x_{t+1} &= f_\theta(L) \cdot x_t = f_\theta(U \cdot \lambda \cdot U^T) \cdot x_t = U \cdot f_\theta(\lambda) \cdot U^T \cdot x = U \cdot f_\theta(\lambda) \cdot \hat{x}_t \\
f_\theta(\lambda) &= \sum_{k=0}^{K-1} \theta_k L^k = \sum_{k=0}^{K-1} U \cdot \theta_k \lambda^k \cdot U^T = U \cdot (\sum_{k=0}^{K-1} \theta_k \lambda^k) \cdot U^T \\
K = 2 &\Rightarrow f_\theta(\lambda) = \theta_0(U \cdot U^T) + \theta_1(U \cdot \lambda \cdot U^T) \\
\theta = \theta_0 = \theta_1 &\Rightarrow f_\theta(\lambda) = \theta(U \cdot U^T + U \cdot \lambda \cdot U^T) \\
x_{t+1} &= (U \cdot U^T + U \cdot \lambda \cdot U^T) \cdot x_t \cdot \theta
\end{aligned}
\tag{4.13}
$$

SpectralCF adds a non-linear transformation, $\sigma$ to the signals, resulting in a non-linear model in Equation 4.14.

$$x_{t+1} = \sigma(U \cdot U^T + U \cdot \lambda \cdot U^T) \cdot x_t \cdot \theta = sp(x_t; \theta) \tag{4.14}$$

Finally, SpectralCF adds $L$ additional layers of depth to the transformation, resulting in the final SpectralCF model in Equation 4.15.

$$x_{t+1} = sp(...sp(sp(x_t; \theta_0); \theta_1)...; \theta_{L-1}) \tag{4.15}$$

Let $T$ be the final timestep for updating. The final embeddings, $x_T$ can simply be a concatenation of the final time step's embeddings at every layer.

$$
\begin{aligned}
x_{T_0} &= x_T \\
x_{T_1} &= sp(x_{T_0}; \theta_0) \\
&... \\
x_{T_L} &= sp(x_{T t_{L-1}}; \theta_{L-1}) \\
x_T &= [x_{T_0}, x_{T_1}, ..., x_{T_L}]
\end{aligned}
\tag{4.16}
$$

For prediction, we simply take the dot product of each pair of (user, item) nodes final embeddings by indexing into $x_T$.

$$\hat{r}_{u,i} = x_T[u] \cdot x_T[i] \tag{4.17}$$

The model is trained with BPR ranking loss [67] using backpropagation. This model is not scalable if it uses all $|V|$ eigenvalues. Hence, we approximate $U$ and $\lambda$ using the largest 100 eigenvalues with their corresponding eigenvectors. This speed up both the model and enables the computation of eigenvalues and eigenvectors with sparse matrices. Note that we can use all 100 frequency filters.

#### Graph Convolutional Matrix Completion (GCMC) [9]

GCMC is the most popular instance of Spatial Graph Convolution for recommender systems. It is more scalable as it does not require computation of eigenvalues and eigenvectors, but is limited as a first-order approximation to full frequency filtering in the Spectral Domain. It only uses the first 2 frequency filters instead of 100 as in SpectralCF. GCMC uses a spatial graph convolution operator as its encoder, then decode using a bilinear function. It is trained using an unsupervised autoencoder method to try and reconstruct its input. **Graph Convolutional Encoder:** Let $C$ be the different rating values and $R_c$ be the binary matrix with value 1 at entries where the rating matrix $R$ has value $c$ and 0 otherwise. Let $D$ be the degree matrix of the bipartite graph and $\sigma$ be the non-linear activation function. Let $W_c$ be

the weights for ratings with value $c$. Then, the **graph convolutional encoder** is defined in Equation 4.18.

$$x_{t+1} = \sigma(\sum_{c \in C} D^{-1} \cdot R_c \cdot x_t \cdot W_c^T) \tag{4.18}$$

**Bilinear Decoder:** Let $Q_c$ be the trained weights for the decoder for ratings with value $c$. The **bilinear decoder** is defined in Equation 4.19.

$$P(R_{u,i} = c) = \frac{e^{(W_u^U)^T \cdot Q_c \cdot W_i^I}}{\sum_{s \in C} e^{(W_u^U)^T \cdot Q_s \cdot W_i^I}} \tag{4.19}$$

**Weight Sharing:** GCMC also employs weight sharing across different rating values for better generalization. For the encoder, $W_q$ is defined as the summation of shareable trained weights $T_s$ as defined in Equation 4.20. For the decoder, $Q_c$ is defined to be the summation of shared weights $V_s$ as defined in Equation 4.21

$$W_c = \sum_{s=1}^{c} T_s \tag{4.20}$$

$$Q_c = \sum_{s=1}^{c} V_s \tag{4.21}$$

**Explicit Feeback:** GCMC is designed to be trained on explicit feedback with ordinal values such as ratings instead of implicit feedback. Hence, we train GCMC on ratings but evaluate it on binarized ratings. We could alter GCMC to also be trained using BPR loss for implicit data but we did not do so. It is the only model which was trained using the original ratings without binarization.

### 4.2.2 Generalizing SkipGram to Recommender Systems

The goal is to generalize the successful use of SGNS in the natural language domain to the recommender domain to learn network embeddings. This section leads to the derivation of **DeepWalk**, our third Deep Graph Embedding model used throughout this thesis. **ProfitWalk** is also a SGNS-based model, which biases the training of DeepWalk towards long-tail items and is introduced in Chapter 7. We briefly mention it here, as it is used throughout this thesis.

We start by reviewing **SGNS** from natural language domains. Then, we review **Prod2Vec** which applies **SGNS** to the rating matrix. Then, we review **LINE** which generalizes **SGNS** to bipartite graphs. However, **LINE** can be viewed as a Breadth-First Search (BFS) model. Hence, we review **DeepWalk** that is Depth First Search (DFS) model on bipartite graph. We also cover **Node2Vec**, which learns to trade-off between BFS and DFS models, but it is not used due to its large computational complexity that does not scale to recommender datasets.

#### SGNS for Natural Language domains

**Skip Gram Negative Sampling (SGNS):** Given a corpus of text, SGNS aims to predict the context of each word, whereby the context of a center word is defined to be words surrounding that center word. For instance, the words that are in the same sentence/paragraph/document of a given center word $\max(P(w_{context}|w_{center}))$. The window context size, $c$ is how far away each surrounding word can

be, and $T$ is the length of each sentence, $w_t$ represents the current center word. Let $|V|$ to be the size of the vocabulary.

$$\underset{\theta}{\text{argmax}}(\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} log(P(w_{t+j}|w_t;\theta))) \tag{4.22}$$

where the probability of context given center word is defined as

$$P(w_{context}|w_{center};\theta) = \frac{exp(\theta^T_{context}\cdot\theta_{center})}{\sum_{k=1}^{|V|} exp(\theta^T_k\cdot\theta_{center})} \tag{4.23}$$

[45] has shown that SGNS inherently factorizes the word co-occurrence matrix in an online manner. The word co-occurrence matrix is a matrix whereby its entries are the number of times each pair of words co-occur together, whereby, we ignore the diagonal entries as it's simply the frequency of each word in the corpus. It is a symmetric matrix of size $|V| \times |V|$ since each word appears in both the row and the column. We can also view the sampling of context and center words as sampling each row from a document-word matrix of size $|D| \times |V|$ where $|D|$ is the number of documents and the entries are either 1 or 0 to indicate the presence of each word in each document. These samples can be thought of as training instances of co-occurring words where we only train the word embeddings without any document embeddings.

#### Generalize SGNS for Rating Matrix

**Prod2Vec: [27]** To generalize SGNS to the recommender domain, **Prod2Vec** view users as documents and items as words in a user-item matrix. Then, it samples the co-occurring item of each user's purchases to train only the item embeddings without training any user embedding. There are a few drawbacks of Prod2Vec.

- Not-personalized. One could also train user nodes by sampling the columns instead of the rows. However, user nodes and item nodes will never co-occur. Hence, we can never define a similarity metric between user and item directly. Hence, [27] only showed that the item embeddings learned is similar to its neighbouring items using user studies but was not able to show that it showed competitive recommendation performance. Our experiments showed that **Prod2Vec** heavily underperformed in standard recommendation metrics. Thus, we have omitted it from this thesis to prevent clutter in experimental results.

- Lack of robustness towards sparsity in the user-item matrix. Items that never co-occur in any user purchases will never be trained to be similar to each other. As the user-item matrix is indeed sparse, directly training on Prod2vec would not be effective.

#### Generalize SGNS for Bipartite Graph

**Prod2Vec** generalizes SGNS into a recommender matrix. However, it would be more useful to generalize SGNS into a recommender bipartite graph. This way, there is no more notion of row or column in a graph. We are also able to see deeper connections between nodes in a graph by traversing the edges in a graph. In this perspective, nodes are said to co-occur if they belong in the same row of the graph adjacency matrix, $A$. **LINE: [74]** overcomes the personalization problem by proposing to preserve both first-order and second-order co-occurrences in a graph. LINE argues that Prod2Vec only preserves second-order

information as it defines co-occurrences of nodes as nodes that share similar neighbour nodes. For instance, item nodes that are connected to the same user node are said to co-occur. However, LINE also defines co-occurrences for first-order nodes, which are user and item nodes that are directly connected to each other. Therefore, user nodes will now be trained to be more similar towards item nodes they are directly connected to and can be used for recommendation tasks.

Using only first-order and second-order information does not overcome the problem of sparsity in recommender bipartite graph. **DeepWalk [64]** overcomes the sparsity problem by proposing higher-order co-occurrences in a graph via random walks in a graph. Thus, in the perspective of SGNS, nodes are words and generated random walks are sentences. This way, the generated random walks are not limited to only first-order, or second-order information and does perform competitively in the recommender domain.

**Node2Vec: [28]** points out that LINE performs Breadth First Search whereas DeepWalk performs Depth First Search when sampling context nodes for each center node. Thus, it proposes to introduce hyperparameters that alter the random walk generation process to learn to trade-off between BFS and DFS generation of nodes and can be viewed as a generalization of Node2vec. However, it requires storing 2nd order sampling information during generation, which is not scalable for recommender datasets as it consumes a space complexity of $O(|V|^2)$. Hence, we only evaluate our experiments on DeepWalk.

To conclude, SGNS applied to recommender dataset is done by first generating context nodes for each node in a graph using different approaches such as second order only as in Prod2Vec, first-order and second-order as in LINE, or arbitrary higher-order as in DeepWalk. Therefore, we call these algorithms DeepWalk in our experiments as only a minor tweak in code translates one approach to another.

### 4.2.3 Possible Extensions

It is possible to extend the architecture for Deep Graph Embeddings:

- **Separate user and item nodes:** We could treat the user and item nodes as different type of nodes during training and evaluation. However, we did not do so in this thesis for simplicity. Removing the user nodes entirely is not advisable as our experiments showed that doing so causes the models to heavily underperform in all evaluations.
- **Different edge types:** We could define different types of edges for different types of implicit feedback such as clicks and purchases. Purchases should contribute more in prediction compared to user clicks. This is superior to Factorization Machines [66] as it overcomes sparsity in the dataset by combing all edge types into a single graph, whereas a multi-dimensional tensor would have extremely sparse tensor in each dimension.
- **Side Information as Node Properties:** We could incorporate side information as additional node properties in the graph.

# Chapter 5

# Analyzing Embeddings of Recommender Systems

In the previous chapter, we looked into the deep embeddings, which we will analyze in this chapter. In this chapter, we aim to understand and analyze the item embeddings for graph recommenders. The item similarity matrix formed using item embeddings is used to analyze these embeddings from numerous recommendation perspectives. The similarity matrix needs to respect metric spaces to make sense from these perspectives.

Section 5.1 covers background material on the recommendation's item similarity matrix, the approaches for computing the item similarity matrix, and the properties of metric spaces. These sections can be skipped for those already familiar with them.

In Section 5.2, we describe the various perspective formed using the similarity matrix in recommender systems that can be used to analyze the embeddings. In addition, we explain why it is important for the item embeddings to respect metric spaces. Next, we contribute methods to analyze these embeddings in Section 5.3. The results from our experiments are located in Section 5.4. Then, we analyze the results from our experiments in Section 5.5 and conclude in Section 5.6.

## 5.1 Background for Analyzing Deep Graph Embeddings

### 5.1.1 Item Similarity Matrix

Item similarity matrix is a matrix that measures the similarity of each pair of items. Let $S^I$: $|I| \times |I|$ denote the item similarity matrix. The entries of this matrix, $S^I_{i,j}$ represents the pairwise similarity score between item $i$ and item $j$.

**Recommender System Item-Item Similarity Matrix**

- $S^I$: $|I| \times |I|$ is the item similarity matrix. The entries of this matrix, $S^I_{i,j}$ represents the pairwise similarity score between item $i$ and item $j$.
- $W^U$ represents the weights for all users
- $W^I$ represents the weights for all items
- $d(I_i, I_j)$ be the distance between items $i$ and $j$.

### 5.1.2 Computing Similarity Matrix

Here, we discuss how existing recommender algorithms compute its item similarity matrix, $S^I$.

**Popularity** simply compute similarities based on the inverse distance of their popularities.

$$S^I_{i,j} = \frac{1}{|r_{:,i}| - |r_{:,j}|} \tag{5.1}$$

**Nearest neighbour** approaches [48] computes similarity score based on item co-occurrences in the rating matrix.

$$S^I = R^T \cdot R \tag{5.2}$$

**Linear recommenders** such as SLIM [60] and LREC [69] computes similarity score by directly optimizing on learning a linear projection of the similarity matrix. Only in this approach, the learnt similarity matrix is not symmetric. To ensure symmetry, we could use equations 5.5, 5.4 for this model instead.

$$\hat{R} = R \cdot S^I = R \cdot W^I \tag{5.3}$$

**Encoder-Decoder** Most approaches in recommender systems can be viewed as an encoder decoder. Then, we can represent the item embedding, $W^I$ as the model's final layer encoded embedding before any decoding. After fetching, the models can indirectly compute the similarity matrix, there are two ways to compute $S^I$.

- Inverse of the distance between the item embeddings.

$$S^I = \frac{1}{|W^I - (W^I)^T|} \tag{5.4}$$

  Inverse norm in equation 5.4 always satisfy **Identity**, **Non-negativity** and **Symmetry**. However, it is not used during training for most models and therefore would not carry any understanding in similarity.

- Dot product within the item embeddings themselves.

$$S^I = W^I \cdot (W^I)^T \tag{5.5}$$

  Dot product in equation 5.5 is used during training for recommender models and better learns similarity structure in datasets. However, it does not respect metric spaces. To prove via counter example, a vector dot product with itself may not be the largest value as it is dependent on the magnitude of the vectors. This does not respect the **Identity** property of having the closest distance to itself and hence is not a metric space [34].

**Matrix factorization** approaches [35, 67, 18] learns the similarity between user and item pairs directly. It directly embeds the user and item vertices into the same low dimensional space, avoiding the computation of a large item-item similarity matrix.

$$\hat{R} = W^U \cdot (W^I)^T \tag{5.6}$$

**Deep Neural Network** approaches [70, 40] trains an item-based autoencoder. Then, a simple

forward pass on each item, $i$ entry $r_{:,i}$ to the final encoder layer before the decoder fetches the needed item embedding, $W^I$.

**Network Embedding** approaches [64, 74, 28] trains an embedding for each node. We simply take the trained embedding as the item embeddings, $W^I$ to compute similarity using equation 5.5.

**Graph Convolutional** approaches, both GCMC [9] and SpectralCF [85] does a forward pass on the nodes. For GCMC, we simply take the last encoding layer before going through the bilinear decoder as the item embedding. For SpectralCF, we take the concatenation of all depth layers of an item node as its final item embedding, $W^I$.

**Collaborative Metric Learning** [34] proposed a model that directly learns distance metric embeddings, but only evaluated its performance using standard ranking metrics.

Most of these approaches do not satisfy all Metric Spaces Properties. In the next section, we propose metrics for evaluating how closely these recommender algorithms meet metric space assumptions.

### 5.1.3 Properties of Metric Spaces

**Metric Spaces** is a rigorous mathematical study on the notion of distance [63]. We can view our notion of similarity as inversely proportional to the notion of distance studied under Metric Spaces. This means that more similar pairs of items should have smaller distances.

Below are the Metric Spaces properties [41]:

Let $d(I_i, I_j)$ denote distance between items $i$ and $j$.

**Identity:** Distance from an item to itself is the smallest possible value and is unique.

$$d(I_i, I_j) = 0 \iff i = j \tag{5.7}$$

**Non-negativity:** Distance cannot be smaller than the smallest possible value.

$$d(I_i, I_j) \geq 0 \tag{5.8}$$

**Symmetric:** Distance has no direction.

$$d(I_i, I_j) = d(I_j, I_i) \tag{5.9}$$

**Triangle Inequality:** Direct path is the shortest distance.

$$d(I_i, I_j) \leq d(I_i, I_k) + d(I_k, I_j) \tag{5.10}$$

## 5.2 Analyzing Embeddings via Similarity Matrix

### 5.2.1 Recommendation Perspectives with Similarity Matrix

The similarity matrix is used to analyze embeddings for recommender systems in the following ways:

**Debugging:** It can be used for **debugging** training by visualizing the space of item embeddings to ensure that *similar* items are embedded close to each other.

**Item Retrieval:** It is used for **item retrieval** recommendations for users who want to retrieve *similar* items to an item they have liked before. For instance, they can compare similar items before

purchase.

- *Here is a list of items similar to item B*

**Personalization:** It is also used for user **personalized** recommendations items based on *similarities* to all their previous purchases and behavior.

**Interpretability:** It ensures the model's recommendations are **interpretable**.

- *Item A is similar to your previously purchased items B, C, and D.*
- *Users similar to you also purchased item A* (using $S^U$ instead)

**Scalability:** Instead of applying learning algorithms globally on the entire space of items, it can apply learning algorithms locally to each small cluster of items [15]. Clusters can be obtained from clustering *similar* items.

### 5.2.2 Similarity Matrix in Metric Spaces

A huge assumption in these applications is that the similarity matrix respects metric spaces. In other words, we assume that the notion of similarity is consistent with our notion of distance in the real world, whereby similarity is inversely proportional to distance. The cluster similarity scores may not be bounded by $[0.0, 1.0]$. Thus, we can only compare their relative scores across model instead of absolute scores.

Triangle Inequality is important as it ensures:

- **Clustering Coherency:** All items within a cluster are somewhat similar to each other [7].
- **Similarity Propagation:** Learns to generalize that two objects are similar if they are both trained to be similar to a third object [34]. This is also the assumption made in Collaborative Filtering models [48]. That is, these models recommend to a user items that another similar user purchased, whereby the similarity is measured based on the number of items that are purchased by both users.
- **Average Similarity:** Averaging over past observations to get most similar items needs to be a vector that represents the average similarity over all these items. For instance, taking an average of objects with the only one particular label should result in being close to an object with that label. This is done for personalization in recommender models [46, 69].
- **Efficient Clustering:** It is also used to accelerate clustering algorithms in high dimensional spaces [58, 24, 39].

## 5.3 Metric Spaces Metrics for Analyzing Recommender Systems

In this section, we analyze the learned embeddings in their ability to satisfy metric spaces properties described in Section 5.1.3. We used the following notation:

- $I_i^{KNN}$ be the set of $K$ closest item to item $i$.

- $C$ be the set of all possible class labels.
- $C_i$ be the set of class labels that item $i$ belongs to.
- $I_c$ be the set of items that contains class label $c$.

Below, we come up with 6 different analysis metrics for metric spaces, each covering a different subset of metric spaces properties.

### 5.3.1 Unsupervised Similarity Analysis

Unsupervised analyses using only the model's item embeddings.

**Identity Similarity:** An item should be most similar to itself.

$$Identity\ Similarity = \frac{1}{|I|}\sum_{i=1}^{|I|} 1[\underset{j}{\text{argmax}}(W_i^I \cdot W_j^I) == i] \tag{5.11}$$

**Triangular Similarity:** Check if each item respects triangle inequality with its K-Nearest Neighbours.

$$Triangular\ Similarity@K = \frac{1}{|I|}\sum_{i=1}^{|I|} \frac{2}{K(K-1)} \sum_{j=1}^{K-1}\sum_{l=k+1}^{K} 1[|W_i^I - W_l^I| \leq |W_i^I - W_k^I| + |W_k^I - W_l^I|] \tag{5.12}$$

where $I_j, I_k \in I_i^{KNN}$

**Cluster Similarity:** Average of similarity within a cluster centered at an item should be similar relative to that item's similarity against its K nearest neighbours similarity score. Hence, a score closer to 1.0 is better. If the value is much lower than 1.0, it means that the neighbours are not similar to each other at all. If the value is much larger than 1.0, this means there exist a niche set of items that is close to every other item, which does not represent a good cluster. This means that the $within_i@K$ similarity is always large as it consists of the same set of niche items for every item.

For each item $i$, take $K$ closest to represent the cluster centered at that item, $I_i^{KNN}$. Then, calculate the average of pairwise similarity within those $K$ items.

The denominator, $center_i@K$ is needed to normalize the score. Without normalization, the score would only be useful for comparison of different clusters within the same model. Therefore, we need to normalize the closeness of items within the cluster to compare different model's metric spaces.

$$\begin{aligned} Cluster\ Similarity@K &= \frac{1}{|I|}\sum_{i=1}^{|I|} \frac{within_i@K}{center_i@K} \\ within_i@K &= \frac{2}{K(K-1)} \sum_{j=1}^{K-1}\sum_{l=j+1}^{K} W_j^I \cdot W_l^I \\ center_i@K &= \frac{1}{K}\sum_{j=1}^{K} W_i^I \cdot W_j^I \\ I_j, I_k &\in I_i^{KNN} \end{aligned} \tag{5.13}$$

### 5.3.2 Supervised Similarity Analysis

In order to further analyze the learned embeddings on metric space properties, we use additional side information available from recommender dataset to analyze the item embeddings. The learned item embeddings are trained in an unsupervised fashion without any knowledge of these side information. Using this approach, we are essentially assuming there is a huge correlation between the user feedback in the implicit matrix and the class labels of items.

To generalize, we assume each item can belong to more than 1 class. This is also known as Multi-Label [23] in literature.

**Binary Similarity:** K-Nearest neighbour predicts the correct label using Binary Cross Entropy.

$$\begin{aligned} Binary\ Similarity@K &= \frac{1}{|I|}\sum_{i=1}^{|I|}\sum_{c=1}^{|C_i|} P_i(class(i)=c)\ log_2(P_{I_i^{KNN}}(class(i)=c)) \\ P_i(class(i)=c) &= \frac{1+\alpha}{|C_i|(1+\alpha)} \\ P_{I_i^{KNN}}(class(i)=c) &= \frac{1}{K}\sum_{j\in I_i^{KNN}} \frac{1[c\in C_j]+\alpha}{|C_j|(1+\alpha)} \end{aligned} \tag{5.14}$$

whereby $\alpha = 10$ is a positive constant that is needed for numerical stability in the case where there exist a class $c \in C_i$ where none of the neighbours of item $i$ intersects to prevent a $log_2(0) = -\infty$.

**Quadruplet Similarity:** Distance between any two items in the same class is closer than any two items in different classes.

Looping through every possible quadruplet in the dataset would take $O(|I|^4)$ time. Hence, we approximate by looking at the $K$ Nearest Neighbour of each item and calculate the average precision over the intersection of class labels. This means we consider a neighbouring item a hit if it contains a class label that intersects with the current item, $i$'s class labels. Then, we calculate the average precision to account for order.

$$Quadruplet\ Similarity@K = \frac{1}{|I|}\sum_{i=1}^{|I|} ClassAveragePrecision@K_{C_i}(I_i^{KNN}) \tag{5.15}$$

**Average Similarity:** Average of items in the same label should result in being closest to another item in the same label. We analyze by fetching the $K$ nearest items to the average of $K$ items in a class, whereby the fetched items cannot intersect the $K$ items used for averaging. Then, we analyze the precision of those $K$ items, whereby a hit is defined to be an item that is in the current class. We call this precision, $ClassPrecision@K_c$.

$$\begin{aligned} Average\ Similarity@K &= \frac{1}{|C|}\sum_{c\in C} ClassPrecision@K_c(I_c^{KNN}) \\ W_c^I &= \frac{1}{K}\sum_{k=1, i_k\in I_c}^{K} W_{i_k}^I \end{aligned} \tag{5.16}$$

For each model, we sample the same set of $K$ items for each class to calculate $W_c^I$. This ensures a fair comparison across models.

## 5.4 Results

We run experiments to analyze the models' performance on metric spaces on both unsupervised similarity analysis and supervised similarity analysis.

Table 5.1: Identity Similarity Analysis

| Model | IdentitySimilarity |
|---|---|
| **DeepWalk** | **0.9969493593654668** |
| ProfitWalk | 0.9920683343502136 |
| KNN | 0.9286150091519219 |
| WRMF | 0.8425869432580841 |
| BPR | 0.7809640024405126 |
| GC-MC | 0.7559487492373398 |
| PMF | 0.6168395363026236 |
| PureSVD | 0.5247101891397193 |
| DeepRec | 0.07931665649786455 |
| SpectralCF | 0.0018303843807199512 |
| Popular | 0.0006101281269066504 |

Table 5.2: MovieLens-100k Identity Similarity

| Model | IdentitySimilarity |
|---|---|
| **KNN** | **0.9771677086164718** |
| DeepWalk | 0.9747213916825225 |
| ProfitWalk | 0.9567817341668932 |
| WRMF | 0.9141070943191084 |
| PMF | 0.8486001630877956 |
| GC-MC | 0.39657515629247075 |
| PureSVD | 0.2304974177765697 |
| BPR | 0.0008154389779831476 |
| DeepRec | 0.0005436259853220984 |
| SpectralCF | 0.0005436259853220984 |
| Popular | 0.0002718129926610492 |

Table 5.3: MovieLens-1m Identity Similarity

| Model | IdentitySimilarity |
|---|---|
| **DeepWalk** | **0.9936** |
| KNN | 0.984 |
| ProfitWalk | 0.9338 |
| WRMF | 0.8808 |
| PMF | 0.8258 |
| GC-MC | 0.7444 |
| PureSVD | 0.181 |
| DeepRec | 0.0502 |
| BPR | 0.0008 |
| SpectralCF | 0.0002 |
| Popular | 0.0002 |

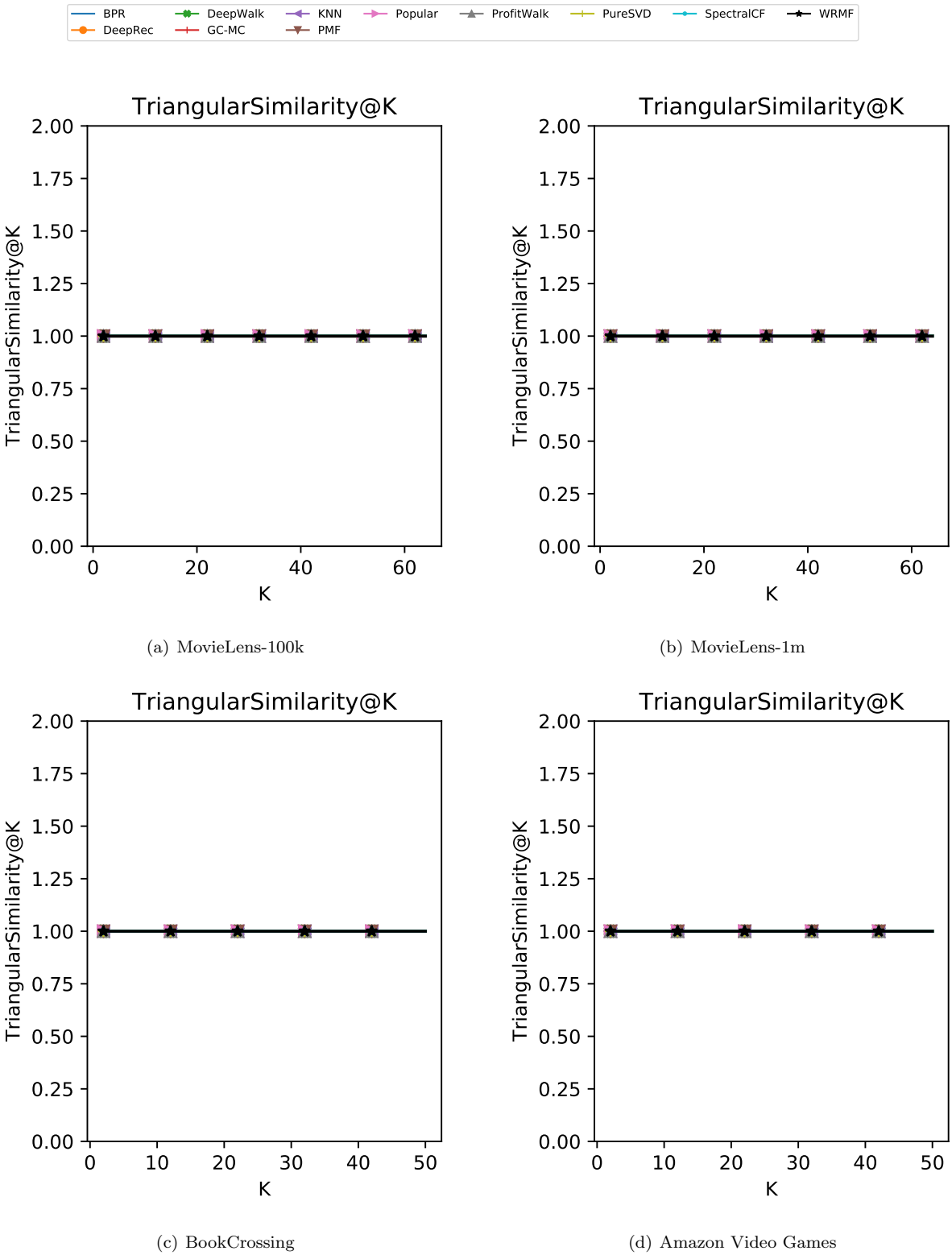
Table 5.4: BookCrossing Identity Similarity

| Model | IdentitySimilarity |
|---|---|
| **DeepWalk** | **0.9959895728895127** |
| KNN | 0.9921796671345499 |
| ProfitWalk | 0.9797473430920393 |
| WRMF | 0.9422498496089834 |
| GC-MC | 0.9049528774814518 |
| PMF | 0.5660717866452777 |
| DeepRec | 0.23180268698616402 |
| PureSVD | 0.09123721676358532 |
| BPR | 0.0010026067776218166 |
| SpectralCF | 0.00020052135552436334 |
| Popular | 0.00020052135552436334 |

Table 5.5: Amazon Video Games Identity Sim.

The recommender models are sorted by their performance.

The **top** model is the best performing model.



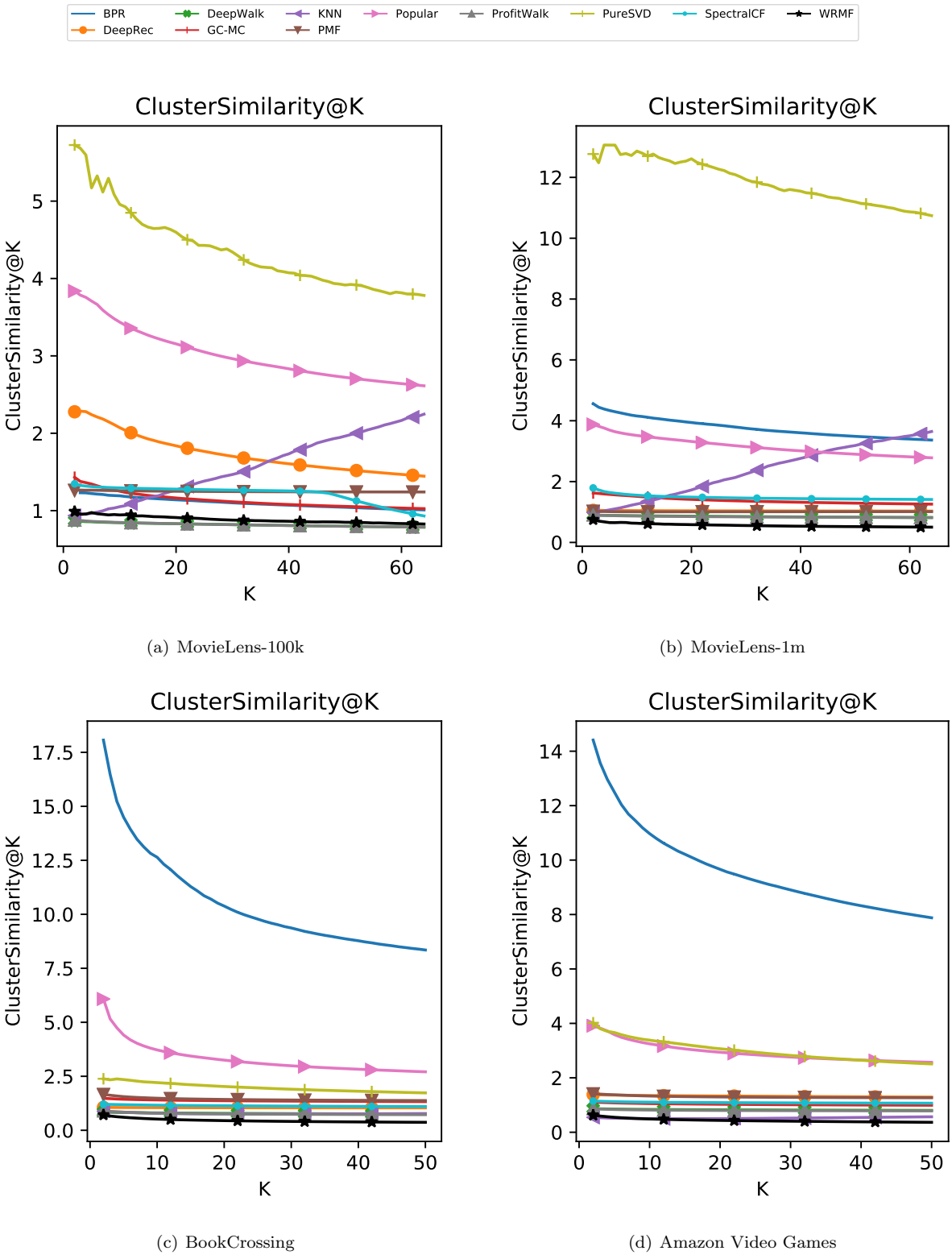

(a) MovieLens-100k

(b) MovieLens-1m

(c) BookCrossing

(d) Amazon Video Games

Figure 5.1: Triangular Similarity Analysis



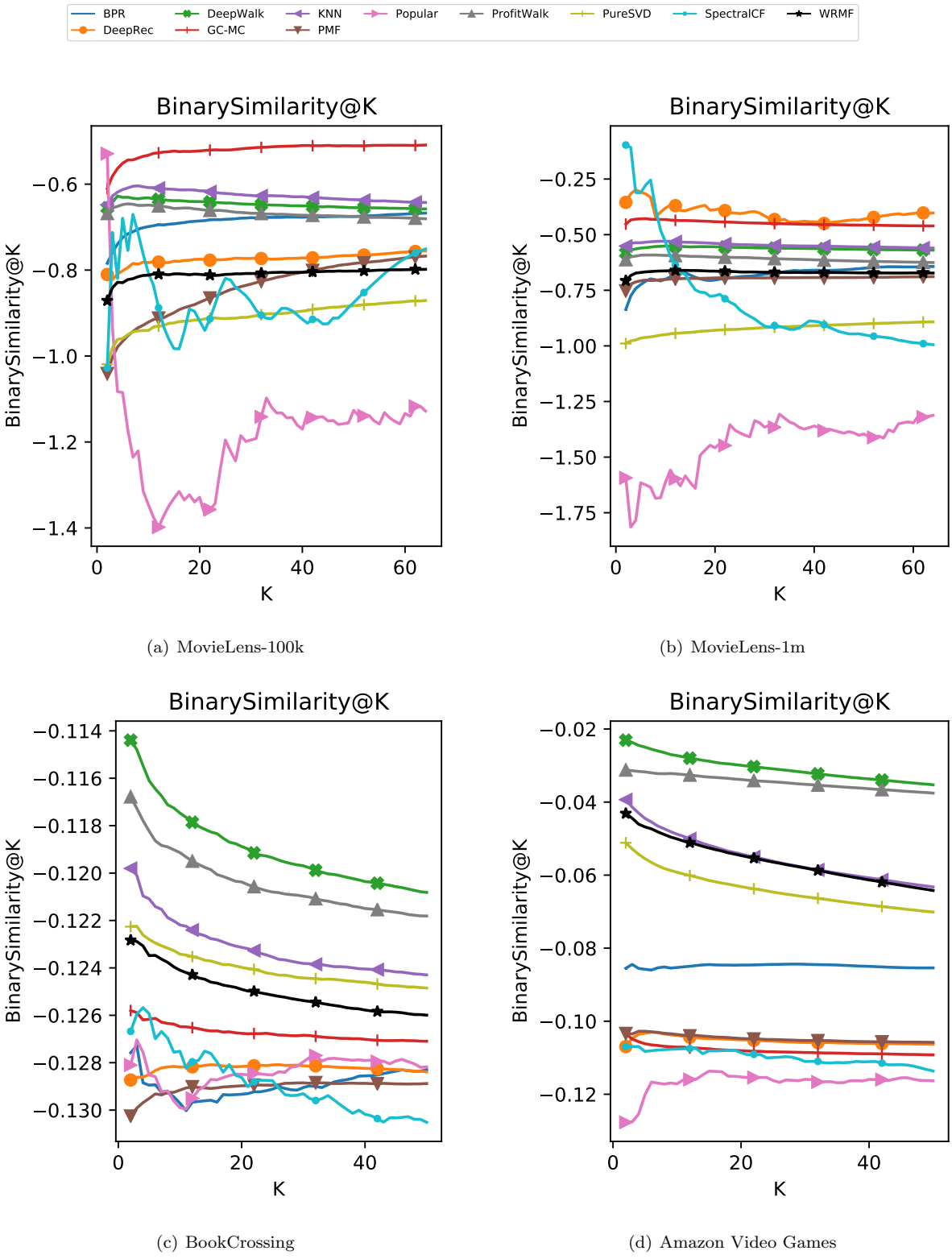

(a) MovieLens-100k

(b) MovieLens-1m

(c) BookCrossing

(d) Amazon Video Games

Figure 5.2: Cluster Similarity Analysis



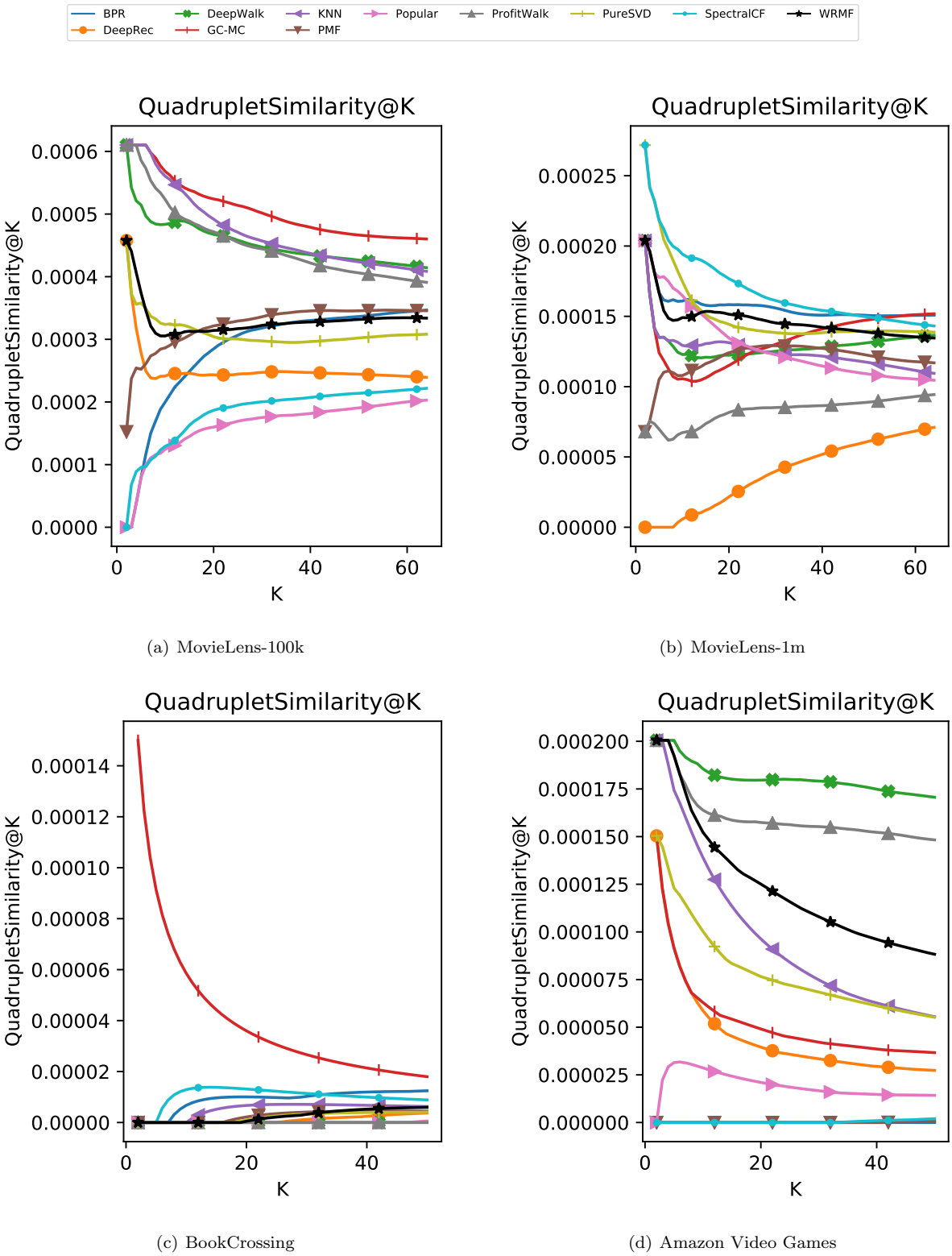

(a) MovieLens-100k

(b) MovieLens-1m

(c) BookCrossing

(d) Amazon Video Games

Figure 5.3: Binary Similarity Analysis




(a) MovieLens-100k

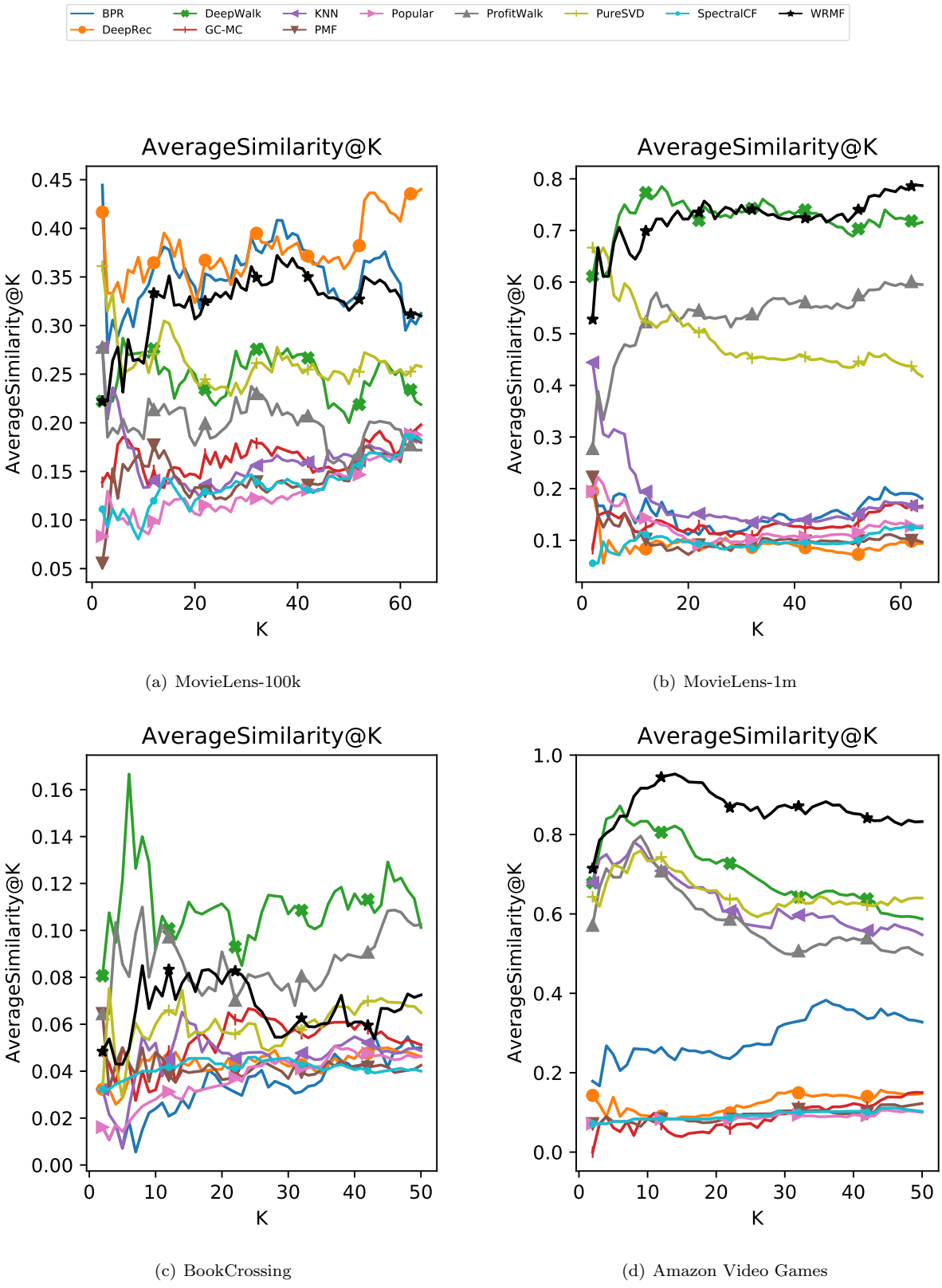
(b) MovieLens-1m




(c) BookCrossing

(d) Amazon Video Games

Figure 5.4: Quadruplet Similarity Analysis




(a) MovieLens-100k

(b) MovieLens-1m

(c) BookCrossing

(d) Amazon Video Games

Figure 5.5: Average Similarity Analysis

## 5.5 Experimental Analysis

### 5.5.1 Identity Similarity Analysis

Referring to Table 5.1, DeepWalk, KNN, ProfitWalk, WRMF, PMF are the best at respecting Identity Similarity whereas PureSVD, DeepRec, SpectralCF, BPR, and Popular does not respect Identity Similarity. Respecting Identity Similarity shows that a model is robust towards large magnitude of popular items when computing similarity.

### 5.5.2 Triangular Similarity Analysis

Referring to Figures 5.1(a), 5.1(b), 5.1(c), 5.1(d) it appears all models respect Triangular Similarity as they are all equal to 1.0.

### 5.5.3 Cluster Similarity Analysis

Referring to Figures 5.2(a), 5.2(b), 5.2(c), 5.2(d), BPR, PureSVD, DeepRec and Popularity are terrible at making sure embeddings that are centered at a certain embedding are similarly close to each other as their ratios are far from 1.0.

DeepWalk, KNN, WRMF, PMF, and ProfitWalk are excellent at doing so as the within item similarity is close in magnitude the centered item similarity. In other words, these models have cluster similarity values that are close to the ratio of 1.0.

Unlike Identity Similarity, SpectralCF performs relatively well under Cluster Similarity.

### 5.5.4 Binary Similarity Analysis

Referring to Figures 5.3(a), 5.3(b), 5.3(c), 5.3(d), DeepWalk, ProfitWalk, KNN, PMF are great at predicting the categorical side information whereas BPR and PureSVD are not. GC-MC underperforms under sparse datasets but surprisingly outperforms on denser datasets. SpectralCF consistently underperforms across all datasets. Being able to predict side information could be useful for modelling information that is not in the implicit feedback matrix.

### 5.5.5 Quadruplet Similarity Analysis

Referring to Figures 5.4(a), 5.4(b), 5.4(c), 5.4(d), GC-MC, KNN, and DeepWalk consistently performs relatively well in Quadruplet Similarity. DeepRec and ProfitWalk consistently perform relatively worse in Quadruplet Similarity. SpectralCF and BPR performs well in some datasets but performs terribly in other datasets, indicating that we cannot draw any conclusion on its performance in Quadruplet Similarity.

### 5.5.6 Average Similarity Analysis

Referring to Figures 5.5(a), 5.5(b), 5.5(c), 5.5(d), DeepWalk, ProfitWalk and WRMF consistently perform relatively well. SpectralCF and GC-MC consistently perform relatively worse. This shows the type of models that we can be personalize simply by averaging over past historical interactions of a user.

## 5.6 Conclusion and Future Work

From the unsupervised metric analysis that does not make use of labeled categorical side information such as Identity Similarity and Cluster Similarity, we can conclude that KNN, DeepWalk, ProfitWalk, WRMF, PMF respects metric spaces more than BPR, DeepRec, PureSVD and Popularity. Hence, models that perform well in Identity Similarity tend to also perform well in Cluster Similarity. However, this is not always true as SpectralCF performs well under Cluster Similarity but underperforms in Identity Similarity.

For supervised metric analysis that do make use of labeled categorical side information such as Binary Similarity, Quadruplet Similarity and Average Similarity, we cannot conclude that a single model is always better as they tend to perform differently among different datasets. We conclude that none of these models are trained to respect metric spaces of categorical side information as there is no model which consistently outperforms other models across all datasets. Therefore, it would only be correct to use a model in ways that assumes it respects metric spaces of categorical side information if it performs well under our proposed metric spaces analysis for the dataset it is being used on. Our inability to draw any conclusions from supervised similarity analysis indicates that our assumption that the implicit feedback matrix highly correlates with categorical item side information is false. The work in this chapter is meant for us to better understand the embeddings, which can help developers debug issues during training, and may not necessarily correlate with standard ranking performance.

In this chapter, we have developed analysis tools to study how well recommender models are trained to respect metric spaces. For future work, it would be useful to design training procedure to guide recommender models to train embeddings that respects metric spaces guided by categorical side information.

# Chapter 6

# Top-$K$ Ranking Recommendation

In the previous chapter, we analyze the graph embeddings from various recommendation perspectives to better understand their properties. In this chapter, we investigate the effectiveness of deep graph embeddings under existing well-known ranking metrics as previously described in chapter 3 by comparing their effectiveness against existing state-of-art recommender embeddings. We also compare their robustness towards sparsity, a common challenge in recommender systems as described in chapter 2.

In Section 6.1, we list the algorithms and datasets used for comparison analysis. The results from our experiments are located in Section 6.2. Then, we analyze the results from our experiments in Section 6.3 and conclude in Section 6.4.

## 6.1 Investigating Effectiveness of Graph Embeddings

To investigate the effectiveness of graph embeddings, we analyze their effectiveness via relative comparison against existing state-of-art recommender algorithms.

The graph embeddings that we investigate in this chapter are:

- SpectralCF: Spectral Collaborative Filtering [85]
- GCMC: Graph Convolutional Matrix Completion [9]
- DeepWalk [64]

The state-of-art recommender algorihms we used for comparison analysis are:

- Pop: Popularity
- KNN: K-Nearest Neighbour [48]
- PMF: Probabilistic Matrix Factorization [57]
- WRMF: Weighted Regularized Matrix Factorization [35]
- PureSVD: Pure Singular Value Decomposition [18]
- BPR: Bayesian Personalized Ranking [67]
- DeepRec: Deep AutoEncoder [40, 70]

To compare performance on robustness towards different sparsity, we analyze changes in their relative performance across datasets with different sparsity as shown in Table 6.1.

To compare performance based on the popularity of items, we evaluated along the a range of values of $K$, where larger $K$ indicates less popular items.

Table 6.1: Scale and Sparsity of Recommender Datasets

| Dataset | $\|U\|$ | $\|I\|$ | $\|R\|$ | $\frac{\|R\|}{\|U\|*\|I\|}$ |
|---|---|---|---|---|
| MovieLens-100k | 943 | 1682 | 100000 | $6.30 \times 10^{-2}$ |
| MovieLens-1m | 6040 | 3706 | 1000209 | $4.46 \times 10^{-3}$ |
| BookCrossing | 7721 | 5000 | 253967 | $6.58 \times 10^{-3}$ |
| Amazon Video Games | 7926 | 5000 | 107359 | $2.71 \times 10^{-3}$ |

## 6.2 Results

In our experiments, we stick to evaluation on $R - Precision$ and $mAP@K$ as they suffice for sparse implicit feedback matrix. We also vary $K$ from 1 onwards to a reasonably large number in a plot for completeness. For user's with less than $K$ available test items, we ignore them for that value of $K$. We run experiments to evaluate the models' performance on both $R - Precision$ and $mAP@K$.

It is overfitting to make conclusions solely based on each table or figure individually. Hence, we jointly analyze several figures or tables together to derive our conclusion in the next section, Section 6.3.

Table 6.2: R-Precision Evaluations

| Model | R-Precision |
|---|---|
| **BPR** | **0.22579149237442533** |
| WRMF | 0.1841242282690616 |
| DeepRec | 0.17368522306349038 |
| ProfitWalk | 0.14920530109228317 |
| PureSVD | 0.134121742616618 |
| DeepWalk | 0.13407648848231915 |
| Popular | 0.12942480308774157 |
| PMF | 0.08037262846870519 |
| SpectralCF | 0.04243381758161209 |
| GC-MC | 0.04206175310215691 |
| KNN | 0.028821922164207498 |

Table 6.3: MovieLens-100k R-Precision

| Model | R-Precision |
|---|---|
| **PureSVD** | **0.16712794423094776** |
| ProfitWalk | 0.14874195923762576 |
| DeepWalk | 0.1376663334186439 |
| WRMF | 0.13525588818481504 |
| BPR | 0.125815116748618 |
| Popular | 0.10634943136289311 |
| DeepRec | 0.07818605888941238 |
| SpectralCF | 0.07615882526088245 |
| KNN | 0.053240445295168304 |
| PMF | 0.04351476384548103 |
| GC-MC | 0.016550159020925787 |

Table 6.4: MovieLens-1m R-Precision

| Model | R-Precision |
|---|---|
| **DeepWalk** | **0.03537717620367315** |
| ProfitWalk | 0.028105778250450776 |
| PureSVD | 0.027979193552098422 |
| BPR | 0.013798620384167482 |
| Popular | 0.013788971609372492 |
| WRMF | 0.01528680219247615 |
| DeepRec | 0.004230860304362016 |
| PMF | 0.0029544114180469483 |
| SpectralCF | 0.0013064369976799556 |
| GC-MC | 0.0011663642887627282 |
| KNN | 0.0007282023657656779 |

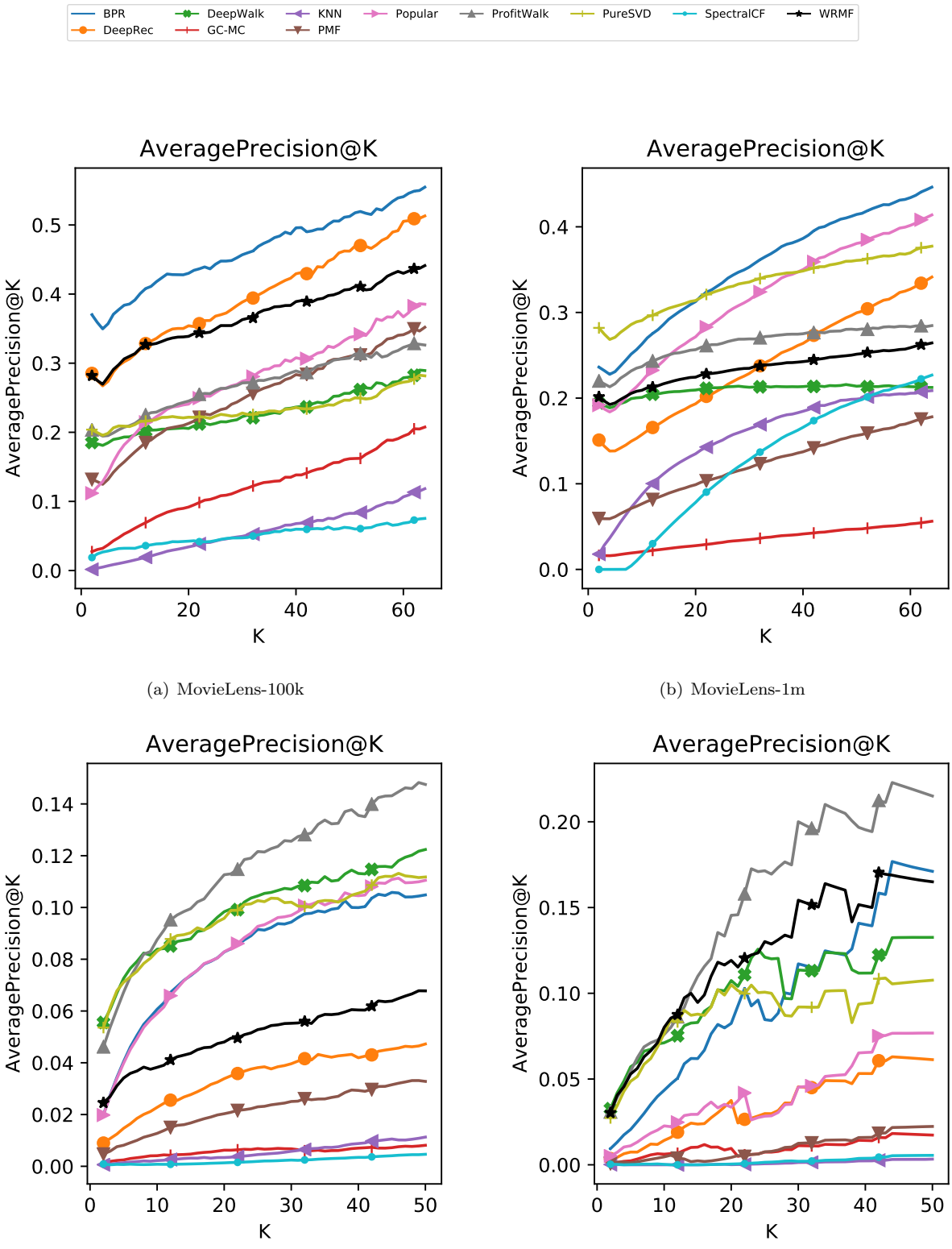
Table 6.5: BookCrossing R-Precision

| Model | R-Precision |
|---|---|
| **DeepWalk** | **0.027476066515407804** |
| ProfitWalk | 0.024723589142794496 |
| WRMF | 0.02267812923406185 |
| PureSVD | 0.01942766456071679 |
| BPR | 0.006751490554507684 |
| DeepRec | 0.0030388102916909794 |
| Popular | 0.00293726167961445 |
| SpectralCF | 0.001033168189966927 |
| GC-MC | 0.0007350850942178361 |
| PMF | 0.0005112025682056364 |
| KNN | 0.0001622652973791078 |

Table 6.6: Amazon Video Games R-Precision

The recommender models are sorted by their performance.

The **top** model is the best performing model.




(a) MovieLens-100k

(b) MovieLens-1m

(c) BookCrossing

(d) Amazon Video Games

Figure 6.1: Mean Average Precision Evaluations

## 6.3 Experimental Analysis

From our experimental analysis as shown in Table 6.2 and Figures 6.1(a), 6.1(b), 6.1(c), 6.1(d), we make the following key observations:

- **Explicit Feedback Performance:** As shown in Table 6.2, recommender systems designed for explicit feedback such as GC-MC, KNN, PMF underperforms in the implicit feedback evaluation metrics. This indicates models designed to be trained under explicit feedback do not perform well under implicit feedback's binarized ranking evaluation metrics.

- **Implicit Feedback Performance:** As shown in Table 6.2 and Figures 6.1(a), 6.1(b), 6.1(c), 6.1(d), classical recommender systems designed for implicit feedback such as BPR, WRMF, PureSVD consistently performs above average across all datasets, also indicating that they are robust towards sparsity.

- **Convolution-based Deep Graph Embedding Effectiveness:** As shown in Table 6.2 and Figures 6.1(a), 6.1(b), 6.1(c), 6.1(d), convolution-based deep graph such as GC-MC and SpectralCF heavily underperforms compared to other recommender models on recommender datasets. This indicates that convolution-based graph embeddings are not suitable to be used as recommender models.

- **Skip-Gram-based Deep Graph Embedding Effectiveness:** As shown in Table 6.2 and Figures 6.1(c), 6.1(d), Skip-Gram-based deep graph such as DeepWalk and ProfitWalk performs competitively compared to existing recommender systems. As shown in the difference in performance between denser datasets in figures 6.1(a) and 6.1(b) compared to sparser datasets in figures 6.1(c) and 6.1(d), DeepWalk and ProfitWalk appear to perform relatively better under sparser datasets, especially on the long-tail.

- **Performance under Sparsity:** Amazon Video Games and BookCrossing are sparser datasets compared to Movielens-100k and Movielens-1m. As shown in Table 6.2 and Figures 6.1(a), 6.1(b), 6.1(c), 6.1(d), the models whose performance relatively improves under sparsity are DeepWalk, ProfitWalk, KNN and SpectralCF.

- **Performance on the long-tail:** As shown in Figures 6.1(a) and 6.1(b), Popular appears to dominate for dense datasets such as Movielens-100k and Movielens-1m at higher values of $K$. However, Popular baseline does not dominate as much in sparser datasets such as BookCrossing and Amazon Video Games as shown in Figures 6.1(c) and 6.1(d).

## 6.4 Conclusion and Future Work

We analyze the effectiveness of deep graph embeddings via relative comparison against existing state-of-art recommender algorithms. Our experiments showed that deep graph models designed for explicit feedback such as GC-MC does not perform well under implicit evaluations. Our experiments also showed that Convolution-based Deep Graph Embedding heavily underperforms compared to existing algorithms. Fortunately, our experiments showed that Skip-Gram-based Deep Graph Embedding appears to be competitive and sometimes outperforms existing seminal recommender systems. The simple popularity

baseline also appears to perform very well on the long-tail items on denser datasets, but underperforms on the long-tail items on sparser datasets. To our surprise, Skip-Gram-based embeddings perform relatively better on sparser datasets, especially on the long-tail items of sparser datasets.

Based on the results of this chapter that showed competitive performance for SGNS-based Deep Graph Embeddings on sparser datasets, especially on the long-tail items, we will proceed to evaluate recommendations on the long-tail items in Chapter 7.

# Chapter 7

# Profitability in Long-Tail Recommendations

In the previous chapter, we compare deep graph embeddings against existing seminal recommendation algorithms on standard ranking metrics. Surprisingly, we learned that the deep graph embeddings excel on the long-tail items although it does not explicitly optimize for them. Long-tail items lead to more profitable recommendations, which is an important goal for firms [81]. Hence, this shows that deep graph embeddings are well suited for profitable recommendations on the long-tail.

In this chapter, we contribute to extend graph embeddings to explicitly optimize for the long-tail items. To better evaluate profitable recommendations, we contribute a new metric that is able to better highlight the difference in performance of recommendations on the long-tail. Then, we analyze and compare the recomendation popularity of different models to analyze the profitablity of their embeddings. From our analysis, we conclude that it could be profitable to bias the training of existing recommender embeddings towards the long-tail items.

Section 7.1 covers background material on popularity, the long-tail, personalization for recommender systems, and evaluation for the long-tail. This section can be skipped for those already familiar with them. In Section 7.2, we contribute by proposing a new metric for evaluating items in the long tail, we call Profitability. We propose an extension to Deep Graph Embeddings, we call ProfitWalk in Section 7.3. The results from our experiments are located in Section 7.4. Then, we analyze the results from our experiments in Section 7.5 and conclude in Section 7.6.

## 7.1 Popularity and Personalization in Recommender Systems

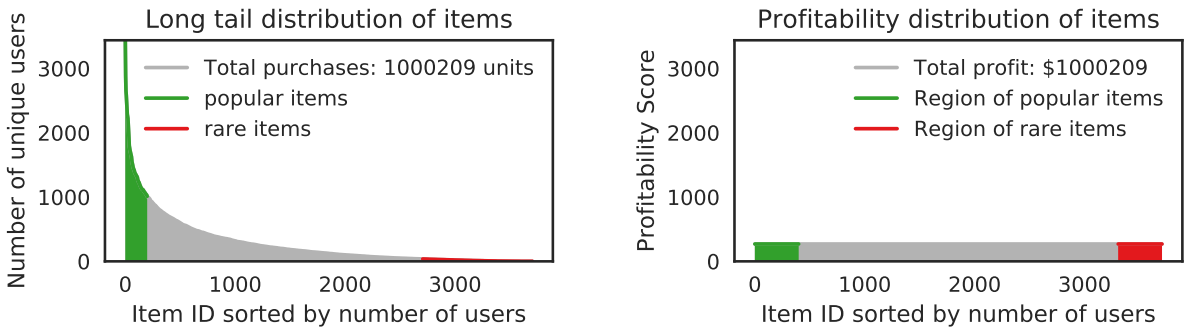
Most recommender systems are known to exhibit popularity bias [73, 4, 83]. Nonetheless, recommending items in the *long tail* can be more profitable [5, 26, 81], personalized [50, 4], and diverse [81]. Existing graph recommenders [62, 81] have been developed for the long tail [62, 81]. However, these algorithms were evaluated using standard IR metrics, which are themselves popularity-biased [8, 12] and may not be the ideal metric for evaluating profitable and personalized recommmendations. In this chapter, we propose an evaluation metric, we coined **Profitability** that directly evaluates the profitability of recommender systems. Our experiments indicate that **Profitability** can highlight relevant profitable and personalized recommendations in its evaluation. To conduct further experiments, we also propose

ProfitWalk, which only differs from DeepWalk [64] in its bias towards long-tail items. Empirically, ProfitWalk outperforms DeepWalk under both standard ranking metrics and Profitability, hinting that it can be useful to bias training of recommender algorithms towards profitable items.

**Definition:**

- **Offline Relevance Evaluation:** Evaluation on models' ability to recommend items that a user did interact with ($r_{u,i} = 1$, known as *hit*)
- **Non-User Personalized Algorithms:** Algorithms that recommends the same set of items for all users.
- **Popularity:** A non-personalized algorithm that strictly prefers items with higher $|r_{:,i}|$.

Below, we show that **Popularity** is the optimal non-user personalized algorithm.




(a) Long Tail Distribution

(b) Profitability Distribution

Figure 7.1: Distribution of Recommender Dataset Movielens Items Before and After Profitability Transformation.

### 7.1.1 Personalization

**Non-User Personalized Algorithm**

The probability of *hit* on an offline dataset when conditioning on an item is $P(hit|item = i) = |r_{:,i}|/|U|$. On an offline dataset, $|U|$ is a normalizing constant, which does not affect optimization. Therefore, the algorithm chooses items based on their respective $|r_{:,i}|$. Referring to Figure 7.1(a)),

- **Popular Region:** Choosing the popular items conditions the distribution towards the high density (green) region of the distribution, which maximizes $|r_{:,i}|$. In this region, it is more likely for recommender models to correctly identify items that are in the test set.
- **Rare Region:** Conditioning on the rare items results in the challenging problem of trying correctly identify the observed test items in the low density (red) region of the dataset. This is because choosing rare items optimizes the probability of missing instead, whereby $rare = \underset{i}{\text{argmin}}(|r_{:,i}|)$.

**Popularity as the best non-user personalized recommender:**
Popularity maximizes precision on a random user,

$$\underset{i}{\text{argmax}}(P(hit|item = i)) = \underset{i}{\text{argmax}}(\frac{|r_{:,i}|}{|U|}) = \underset{i}{\text{argmax}}(|r_{:,i}|) \tag{7.1}$$

Hence, it is the best non-user personalized algorithm (on the train set) and performs well (as shown in Figures 7.3(a) and 7.3(c)).

**Difficulty evaluating personalization**

Any recommendation that performs better than **Popularity** can be said to be more personalized. However, it is difficult to outperform **Popularity** under standard ranking metrics because recommender models would need to learn to predict items favorably in the low density (red) region of the dataset (refer to Fig. 7.1(a)), where **Popularity** will never *hit* (refer to Fig. 7.5). Even if they do so, their performance insignificantly affects standard metrics that naively rely on the recommender test set. The recommender test set is unable to highlight the minor yet important differences between different recommender models as their performance on the low density (red) tail region of the test set are overshadowed by their performance on the high density (green) region of the test set.

Assuming recommender models are indeed popularity biased, they should perform similarly on the popularly bias test set. Our experiments in figures 7.3 and 7.5 support this assumption. Therefore, it would be difficult to evaluate the personalization and profitability of popularity-biased algorithms on standard IR metrics.

### 7.1.2 Evaluating recommendations on the long-tail

Existing recommendation work has tried to increase recommendations for rare items in the long-tail. These work are classified under diversity and novelty. We briefly cover them below and explain how our work differs.

- **Visualization:** Evaluation on diversity for generative models are done based on the diversity of realistically generated data assuming the input can be visualized (e.g. visual and language tasks). However, this approach is impractical in the recommendation space because one cannot visualize the rating matrix. It is simply a rating with a lot of numbers. [70] has tried to evaluate diversity by analyzing a small set of cherry-picked samples from prediction. However, this approach requires user surveys and does not globally compare different model's prediction across the entire item space.

- **Minimize Recommended Similarities:** Metrics such as MMR [13, 79] and ItemNovelty [82] measures the diversity of top-N recommended items by penalizing high similarities between recommended items. However, this requires computing a dense item-item similarity, which is not scalable for sparse and large recommendation datasets. Similarity calculation would also be ineffective due to the sparsity of the recommendation matrix (as shown by poor performance of KNN on Figure 6.1). One could counter sparsity by training a small model to predict item-item similarity. However, this violates evaluation because evaluation must be independent of any specific model that outputs item-item similarity predictions.

- **Different unique items:** [43] evaluated diversity by simply counting the number of different items in recommendations across all user. This is faulty as a random model that merely recommends items uniformly at random would perform the best. This is because the naive approach

measures the entropy of the recommendation distribution, whereby the uniform distribution has maximum entropy. Counting diversity per user does not work as the recommender system could just recommend the top K popular items for all users. Also, this approach also requires computing diversity globally across all users. Hence, it is sensitive to the number of users in a system. Our profit-margin approach allows us to know which users are our most profitable customers.

- **Repiprocal of popularity:** [75] evaluated novelty using a simple reciprocal of popularity, similar to what our math derivation leads to. However, a simple reciprocal is not properly normalized (ideal model which recommends items for every user correctly does not end up with a score of 1).

## 7.2 Profitability

### 7.2.1 Economic Lens

We now view the recommender system problem from an economic perspective to motivate our Profitability metric in Section 7.2.2.

**Profit Margin Perspective:** Each item has different levels of profit margin. Each popular item sale rewards less profit compared to each unpopular item sale. Popular items are widely sold by many firms. Hence, each unit selling price would be the market price, which rewards little profits. Unpopular items are only sold by a few firms. Thus, these few firms earn major profits from each sale as they have less competition and more price-making power [81]. In the extreme case, you can think of these businesses as operating in an oligopolistic market for unpopular items. The reason firms do not sell unpopular items is because they are difficult to sell (red long-tail region of the test set). It is then up to a good recommender system to figure out which of these unpopular items are profitable to their marketed customers.

**Profitability as Inverse Popularity** Viewing from an economic lens, we can assume each item sale produces varying profits. We assume that the profitability of an item is inversely proportional to its popularity.

**Independence of Popularity:** This assumption results in an evaluation metric that is no longer biased towards popular item as shown in Figure 7.1(b). In this metric, the area under the curve of both the green region and red region are now equal. This ensures that the evaluation is independent of the popularity of any item.

**Investment:** Our evaluation adds an investment component into the recommendation problem. To perform well using *Profitability* metric, recommender models will have to intelligently tradeoff between low risk-returns from popular items and high risk-returns from rare items. This can also be viewed as a form of exploration-exploitation trade-off, allowing our evaluation metric to be used as an offline evaluation metric for contextual bandits [84].

### 7.2.2 Profitability Evaluation Metric

We will now formalize our profitability metric. The reason that the IR metrics from Section 3.2.2 are unable to highlight recommendations in the long-tail is that it assigns the same score for correct predictions or *hit* on any item, independent of the item's popularity. Hence, popular items dominate the evaluation score. We overcome this deficiency by assigning a different item-dependent score for each

*hit*. This leads to a distribution score that is uniformly distributed across all items, regardless of their popularity.

We generalize the IR metrics from Section 3.2.2 from assigning a single constant value of 1 for each *hit* to an item-dependent variable, we call evaluation score $s_i$. Hence, the equations below look similar to those from Section 3.2.2, except that they can be viewed as a generalization that uses an item-dependent variable, $s_i$ instead of a single constant value of 1. Hence, you can derive the precision metrics from Section 3.2.2 by simply setting $s_i = 1$. We call this the **Standard Precision**. Instead, setting $S_i = \frac{1}{|r^{test}_{:,i}|} * (\frac{|R^{test}|}{|I^{test}|})$ leads to our proposed metric, we call **Profitability**. **Profitability** is motivated to highlight recommendations in the long-tail while maintaining the total score available from the test set, as shown in Table 7.1.

Below, we generalize the definition of Precision from section 3.2.2 to be defined in terms of $s_i$. For a given user, $u$, let:

$$
\begin{aligned}
\textbf{HitItemSet@}K_u &= \hat{r}_{u,:K} \cap r^{test}_{u,:} \\
\textbf{HitScore@}K_u &= \sum_{i \in \textbf{HitItemSet@}K_u} s_i \\
\textbf{OptimalScore@}K_u &= \sum_{i \in r^{test}_{u,:K}} s_i \\
\textbf{Precision@}K_u &= \frac{\textbf{HitScore@}K_u}{\textbf{OptimalScore@}K_u} \\
\textbf{AP@}K_u &= \textbf{AveragePrecision@}K_u = \frac{1}{K}\sum_{k=1}^{K} \textbf{Precision@}k_u \\
\textbf{mAP@}K &= \textbf{AveragePrecision@}K = \frac{1}{|U^{test}|}\sum_{u \in U^{test}} \textbf{AP@}K_u \\
\textbf{HitScore}_i &= \sum_{u \in r^{test}_{:,i}} s_i = |r^{test}_{:,i}| \times s_i \\
\textbf{TotalScore} &= \sum_{i \in I^{test}} \textbf{HitScore}_i
\end{aligned}
\tag{7.2}
$$

Table 7.1: Comparison between Standard and Profitability

| Properties | Standard | Profitability |
|---|---|---|
| $\textbf{score}_i = s_i$ | 1.0 | $\frac{1}{\lvert r^{test}_{:,i}\rvert} * (\frac{\lvert R^{test}\rvert}{\lvert I^{test}\rvert})$ |
| $\textbf{HitScore@}K_u$ | $\lvert\textbf{HitItemSet@}K_u\rvert$ | as defined above |
| $\textbf{OptimalScore@}K_u$ | $K$ | as defined above |
| $\textbf{HitScore}_i$ | $\lvert r^{test}_{:,i}\rvert$ | $\frac{\lvert R^{test}\rvert}{\lvert I^{test}\rvert} = constant$ |
| **TotalScore** | $\lvert R^{test}\rvert$ | $\lvert R^{test}\rvert$ |
| $range(\textbf{mAP@}K)$ | [0.0, 1.0] | [0.0, 1.0] |

**Properties of Profitability:** Referring to Figure 7.1(b) and Table 7.1, *Profitability* maintains the total score, $\textbf{TotalScore} = |R| = 1000209$ and is independent of popularity, $\textbf{HitScore}_i = constant$.

These useful properties result from only a simple transformation for the definition of $s_i$ on most existing standard evaluation metric. We focus only on standard Precision ($\textbf{mAP@}K$) due to space limitations. Our definition can be thought of as a generalization of precision, whereby standard precision uses $s_i = 1$.

From referring to Table 7.1, we see that a well understood range of $[0, 1]$ and the total score is maintained. This means that we do not create additional score mass, but simply transfer score mass from the popular items to the less popular items. It also means we are robust towards the distribution of item popularity in the dataset. It simply transform any distribution of item popularity to a uniform distribution of evaluation score as shown in Figure 7.1(b).

## 7.3 ProfitWalk Algorithm

In the previous section, we derived the Profitability metric that is able to highlight performance of recommender systems on the long-tail. In this section, we propose to extend deep graph embeddings to explicitly optimize for the long-tail items.

### 7.3.1 Random Walk in Networks

Given that the graph is unweighted without any additional information about each node, we focus on using only the network structure to guide training. We view *links* as a path from walking from a node to another.

#### Random Walk Generation

A list of similar nodes, $\{v_0, v_1, ..., v_t, ..., v_L\}$ is generated by randomly traversing the graph starting at node $v_0$ for $L$ steps. We denote $v_t$ as the $t^{th}$ node in the walk where it is possible to walk backward (i.e. $v_{t-1} = v_{t+1}$). The steps are generated using a transition matrix, $T \in \mathbb{R}^{|V|\times|V|}$, with entries $T_{i,j} = P(v_{t+1} = j | v_t = i)$.

### 7.3.2 ProfitWalk

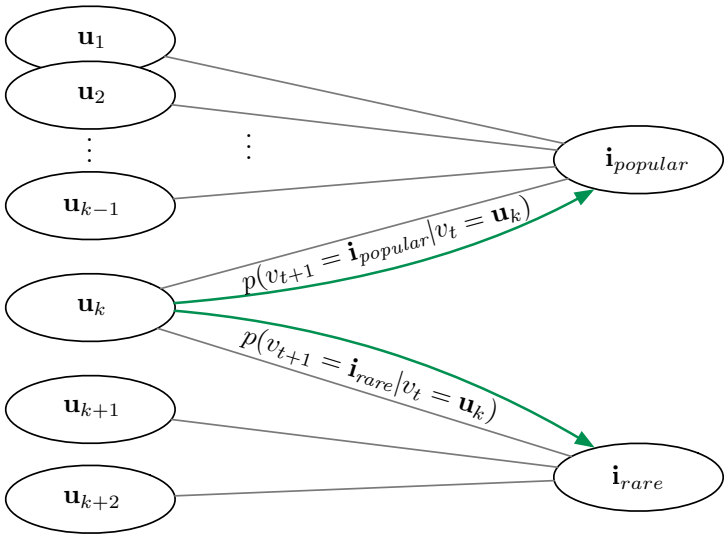



Figure 7.2: Walk from User to Items

**Profitable Walks:** We define similarity as to how ***profitable*** it is to walk from one node to another. Referring to Figure 7.2, user node $u_k$ can either walk towards node $i_{popular}$ or $i_{rare}$. Walking

towards $i_{popular}$ leads to market competition from many other users $\{u_1, u_2, ..., u_{k-1}\}$ who also have a direct path towards $i_{popular}$ whereas walking towards $i_{rare}$ leads to a more profitable item as it has less competition (only $\{u_{k+1}, u_{k+2}\}$). **ProfitWalk** prefers walking from popularity to profitability (i.e., $P(v_{t+1} = i_{rare}|v_t = u_k) > P(v_{t+1} = i_{popular}|v_t = u_k)$).

**Stochastic Walks for Personalized Embeddings:** Random walk based training leads to personalized embeddings. Referring to Figure 7.2, there are many possible nodes to walk to from node $i_{popular}$. Hence, it is unlikely for node $u_k$ to be its chosen path. As a result, $u_k$'s latent embedding does not get closer to $i_{popular}$'s latent embedding. On the other hand, it is very likely to walk from item $i_{rare}$ to node $u_k$. Hence, $u_k$'s embedding is likely to be *personalized* towards node $i_{rare}$. This is empirically shown in Figure 7.5, whereby both DeepWalk and ProfitWalk produces *personalized* recommendations.

**DeepWalk Transition Matrix:** $T_{deep} = D^{-1} \cdot A$

**ProfitWalk Transition Probability:**

$$P(v_{t+1} = j|v_t = i) = \frac{D^{-1}_{j,j} \times A_{i,j}}{\sum_{(i,m)\in E} D^{-1}_{m,m} \times A_{i,m}} \tag{7.3}$$

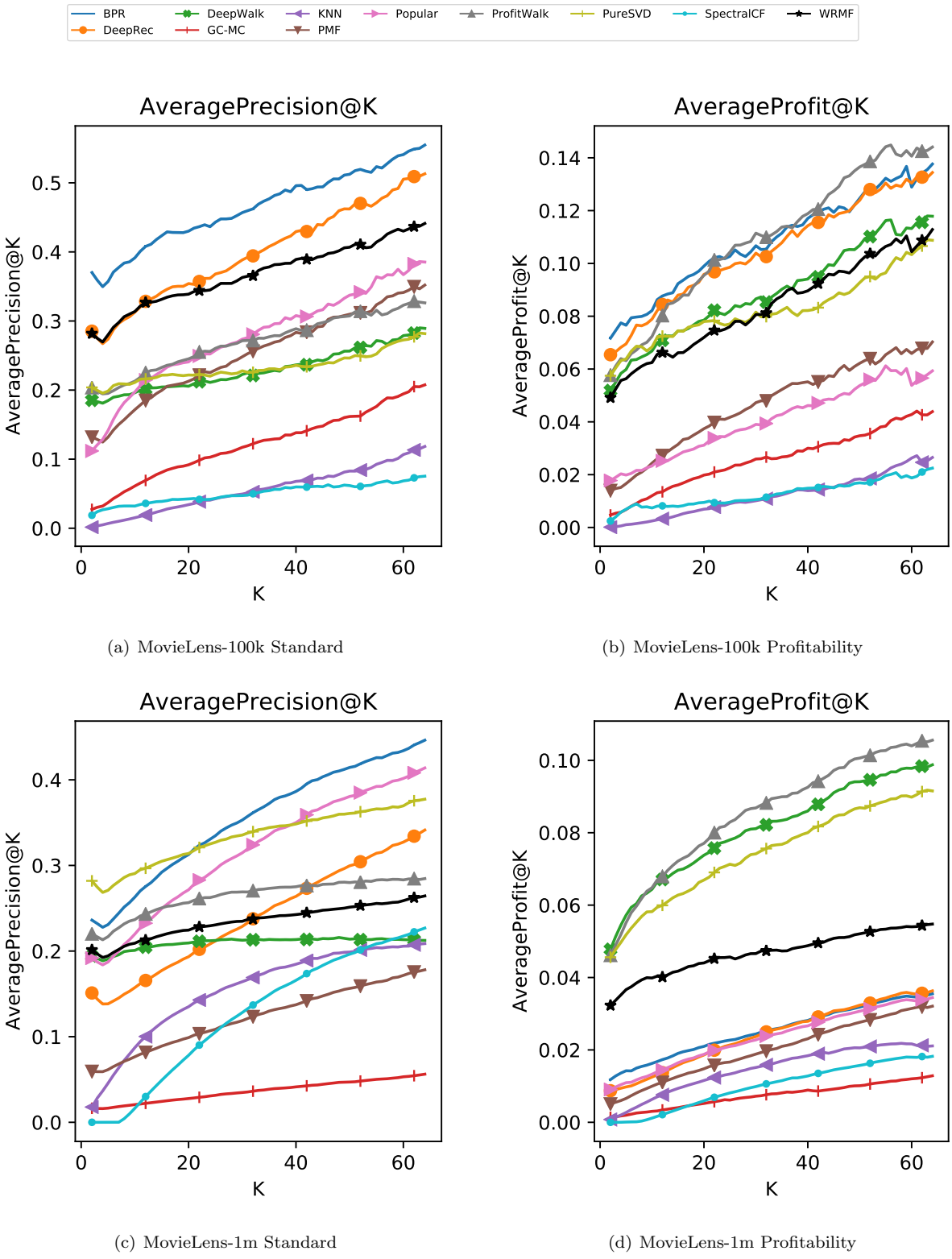
where $D \in \mathbb{R}^{|V|\times|V|}$ is the diagonal degree matrix of $G$.

### 7.3.3 Training ProfitWalk

As in DeepWalk [64], the nodes in each generated walks are trained using **SkipGram** [56, 64, 27]. Within each generated walk, **SkipGram** pulls each node's embedding closer to one another.

## 7.4 Results

We run experiments to evaluate the models' performance on profitability precision compared to standard precision as well as analyze the recommendation popularity of different models.



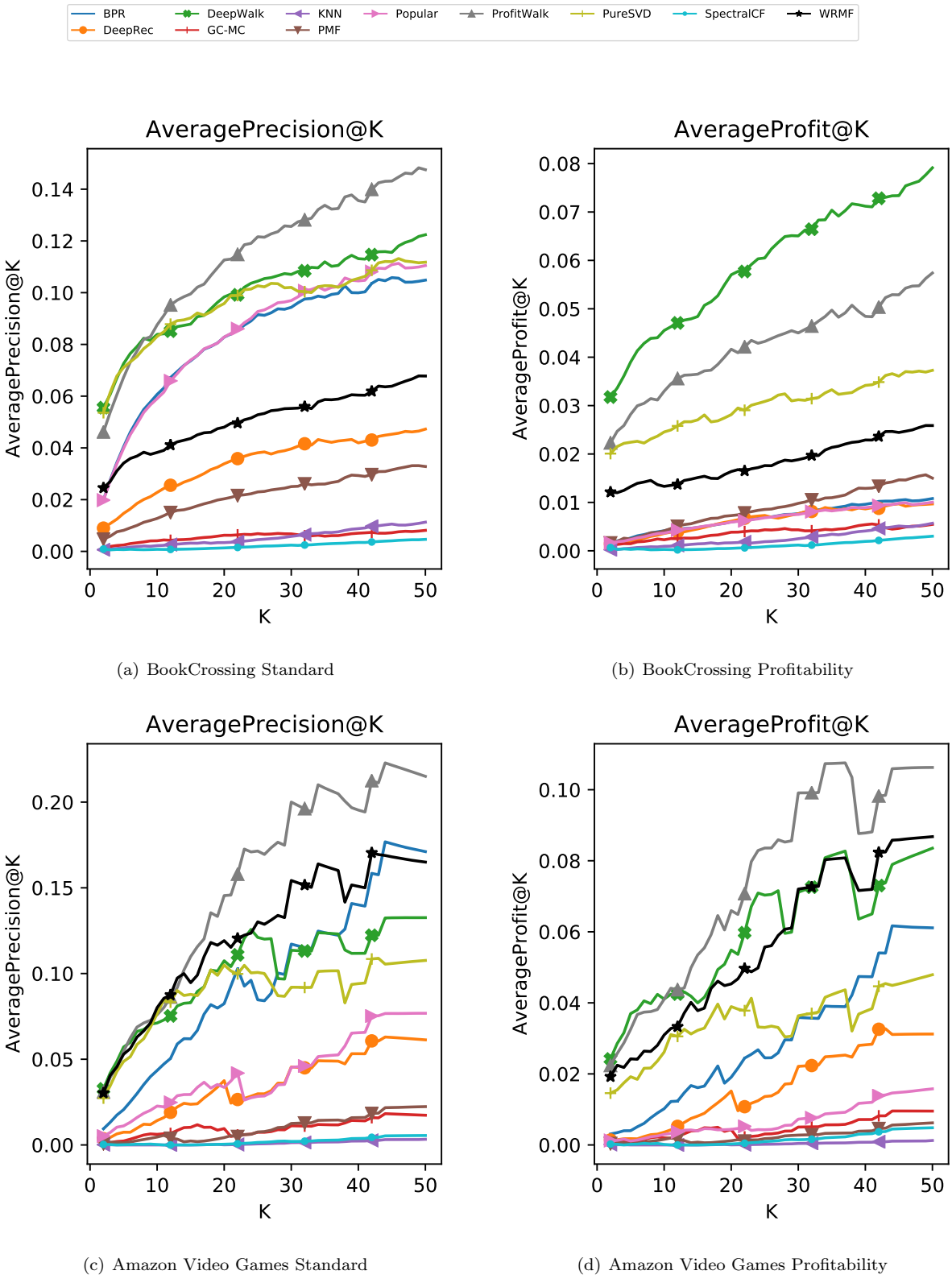

(a) MovieLens-100k Standard

(b) MovieLens-100k Profitability

(c) MovieLens-1m Standard

(d) MovieLens-1m Profitability

Figure 7.3: Standard Precision and Profitability Evaluation of Algorithms on Movielens Datasets.




(a) BookCrossing Standard

(b) BookCrossing Profitability

(c) Amazon Video Games Standard

(d) Amazon Video Games Profitability

Figure 7.4: Standard Precision and Profitability Evaluation of Algorithms on BookCrossing and Amazon Video Games Datasets.

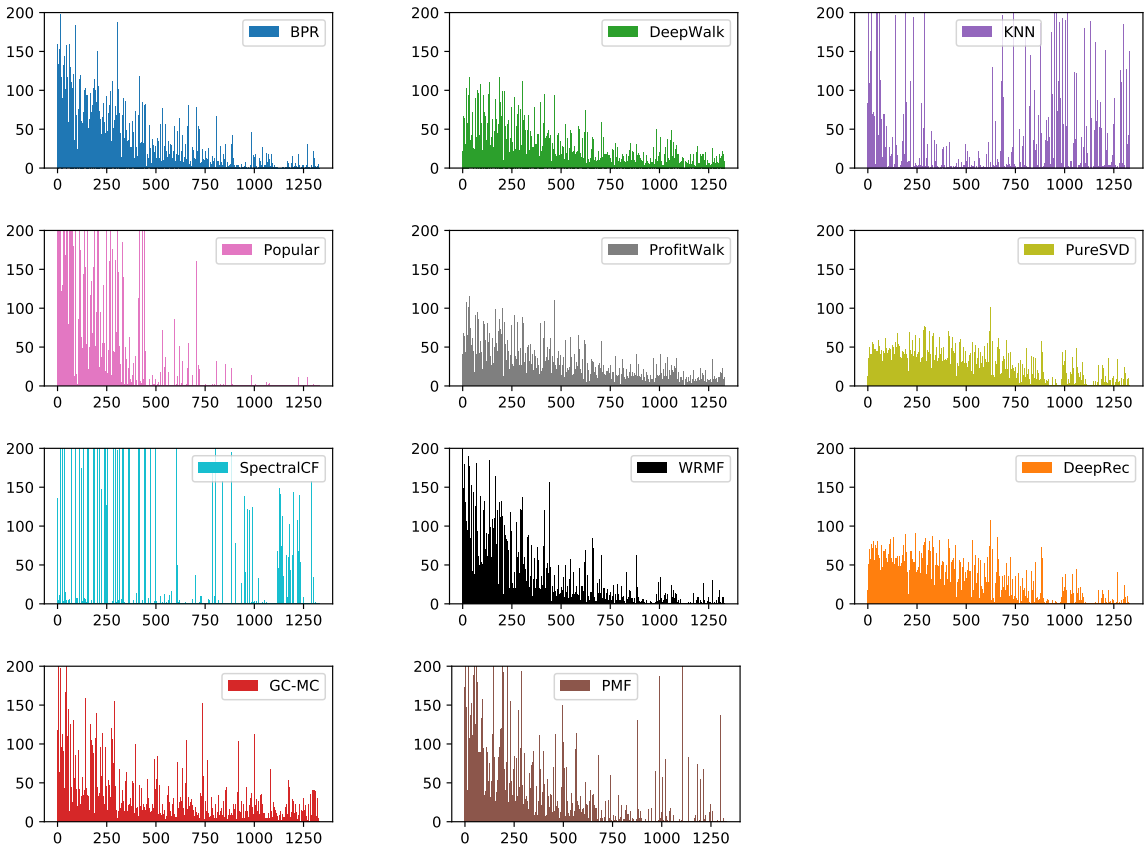



Figure 7.5: Recommendation Popularity of Algorithms on MovieLens-100k

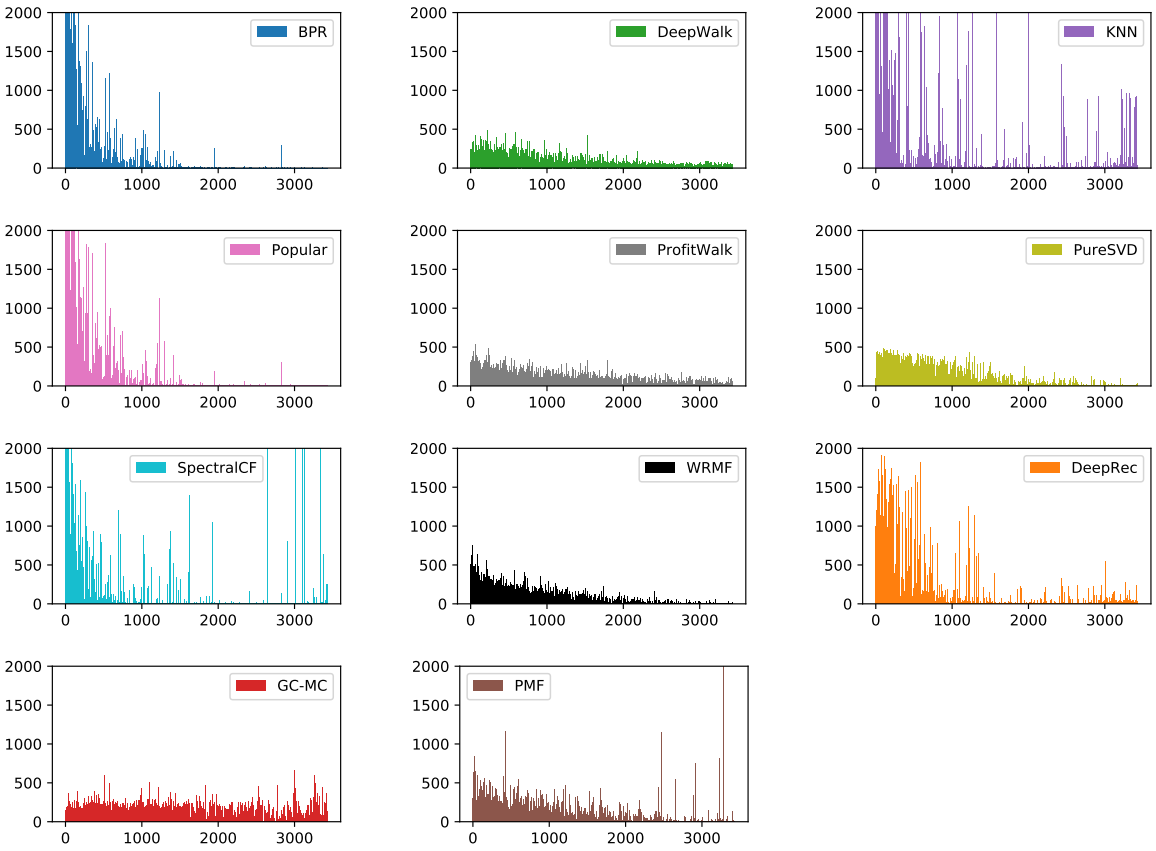



Figure 7.6: Recommendation Popularity of Algorithms on MovieLens-1m

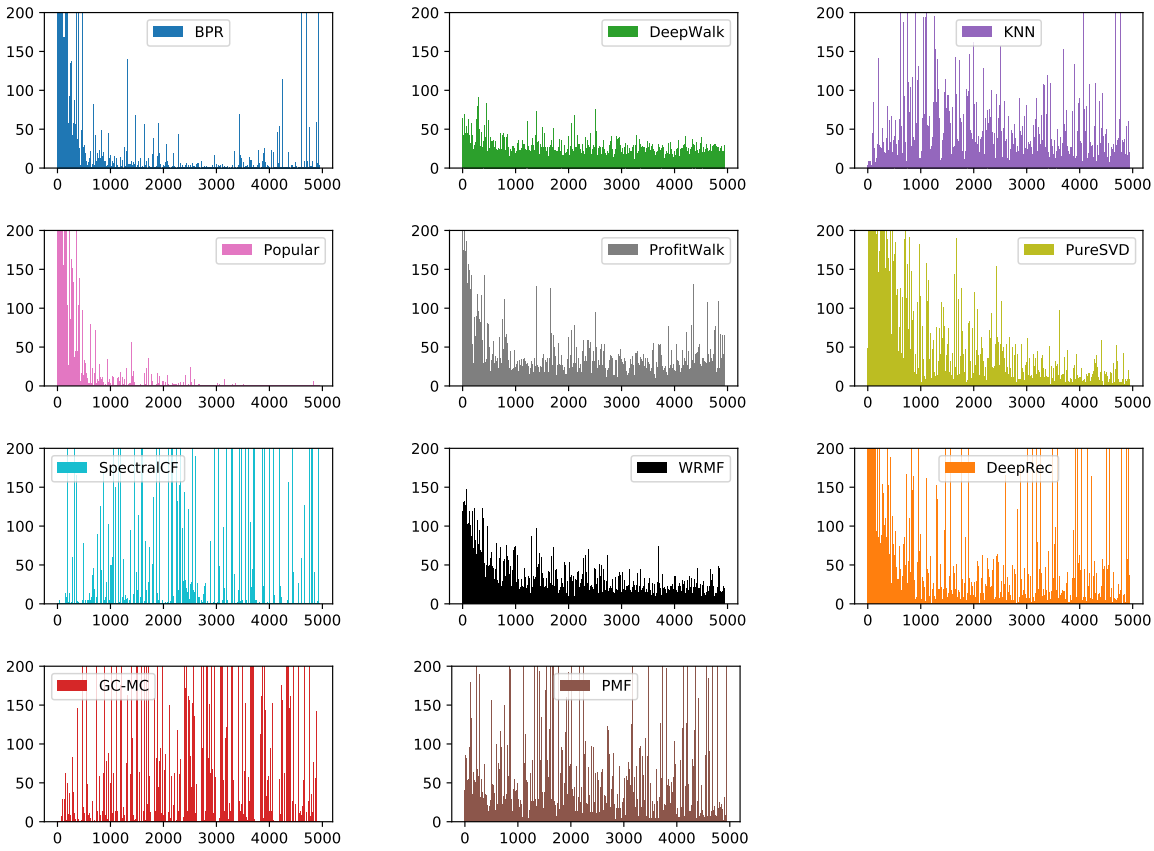



Figure 7.7: Recommendation Popularity of Algorithms on BookCrossing

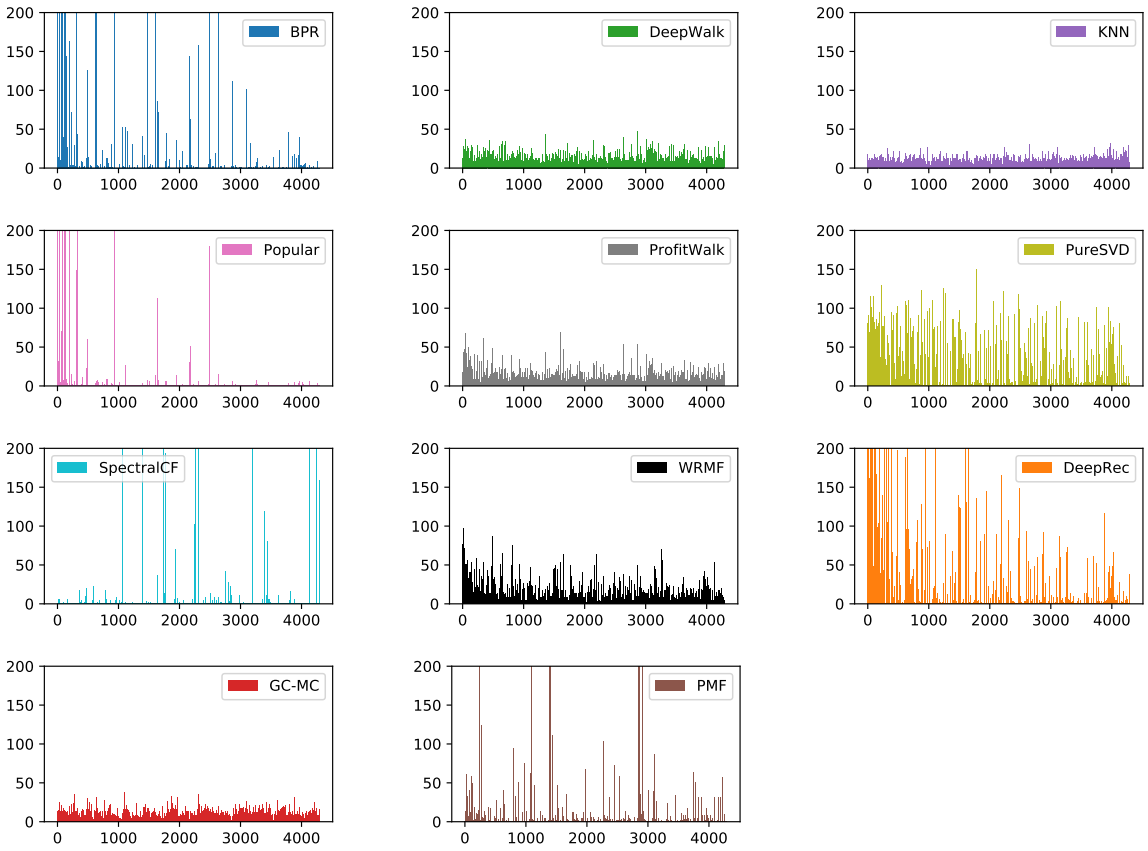



Figure 7.8: Recommendation Popularity of Algorithms on Amazon Video Games

## 7.5 Experimental Analysis

### 7.5.1 Recommendation Popularity of Algorithms

Referring to Figures 7.5, 7.6, 7.7, 7.8, DeepWalk and ProfitWalk appears to be most personalized in their recommendations, whereas DeepRec, KNN, BPR, SpectralCF and Popularity are heavily biased (exceeds 2000 in MovieLens-1m) towards popular items.

### 7.5.2 Connections between Profitability and Diversity

Algorithms that more evenly distributes its recommendation across the entire item spectrum (zooming into Figure 7.5) such as ProfitWalk, DeepWalk, PureSVD appears to perform well on Profitability metric (refer to Figure 7.3(d)), whereas algorithms that are heavily biased towards popular items such as Popular and BPR drastically drops in relative performance under Profitability. For instance, in Figure 7.3(c), it appears that both SpectralCF and Popularity increases with $K$, whereas they no longer increase significantly with $K$ in Figure 7.3(d), indicating that the embeddings of SpectralCF are heavily biased towards popular items.

### 7.5.3 Profitability on Personalization

We analyze non-user-personalize model, **Popular** on Profitability. It turns out the relative performance of **Popular** drops significantly in Profitability evaluation. This implies that models will need to learn to be personalized to perform well under Profitability evaluation.

### 7.5.4 Biasing Training for Profitability

ProfitWalk appears to almost always outperform DeepWalk (in MovieLens-100k, MovieLens-1m, Amazon Video Games) in both standard and profitability precision metrics, indicating that it could be useful to bias training of other models towards profitability. Only in BookCrossing does DeepWalk perform better than ProfitWalk under Profitability, however, it turns out the popularity distribution for DeepWalk's embeddings on this dataset somehow learned to be more evenly distributed compared to ProfitWalk as shown in Figure 7.7. Therefore, we can conclude more evenly distributed embeddings should perform better under Profitability.

## 7.6 Conclusion and Future Work

We propose a new evaluation metric, we call Profitability that is able to highlight performance of recommender systems on the long-tail items. Profitability is designed to be robust towards the distribution of item popularity. Our Profitability metric also encourages development of recommender models that are biased towards profitable recommendations.

We also propose ProfitWalk, an extension of Deep Walk that explicitly walk towards Profitable items. Surprisingly, it performs better than DeepWalk on both the standard IR metrics and our proposed Profitability metric. We showed that DeepWalk could easily be extended by simply altering its transition matrix for random walk during training.

We also analyze the recommendation popularity of embeddings and showed that existing recommender embeddings are biased are towards popular items. We showed that both ProfitWalk and DeepWalk have better popularity distribution compared to other recommender models. From highlighting the connections between popularly biased embeddings and their performance on the long tail items, our experiments indicate that it could be useful to bias recommender models towards profitable items.

For future work, we suggest adapting deep graph embeddings to optimize for additional metrics by simply altering their walks to walk towards those metrics. An example is to be remove sensitive bias that are present in the dataset [10].

# Chapter 8

# Conclusion

## 8.1 Summary of Contributions

The contributions of this thesis to address our research questions are:

**Question 1: What are the properties of graph embeddings?**

- Propose three supervised metrics that use labeled item side information to analyze item embeddings.
- Propose three unsupervised metrics that use mathematical theory to analyze item embeddings.
- Showed that no model consistently respect supervised metric analysis and do not consistently predict categorical side information well, indicating that we should not assume recommender models respect metric spaces when using them for its applications.
- Showed that SGNS-based Deep Graph Embeddings perform well under unsupervised metric analysis.

**Question 2: How do graph embeddings compare to state-of-the-art algorithms?**

- Showed Convolution-based Deep Graph Embeddings underperformed compared to existing recommender systems
- Showed SGNS-based Deep Graph Embeddings are competitive compared to existing recommender systems
- Showed SGNS-based Deep Graph Embeddings are robust towards sparse datasets and performs relatively better compared to other models under sparse datasets, especially for the long-tail items.

**Question 3: How to extend graph embeddings for personalized recommendations?**

- Analyze the intensity of embedding bias towards popular items using Recommendation Popularity of existing models and showed that most classical models are heavily biased towards popular items.
- Propose a new evaluation that can highlight the performance of recommender systems on the long-tail.

- Propose a new model called ProfitWalk that extends DeepWalk to explicitly optimize for items in the long-tail.
- Surprisingly showed that ProfitWalk also performs better compared to DeepWalk under standard IR metrics.

## 8.2 Future Work

Based on our conclusion in this thesis, we propose the following future work:

- Develop embeddings that explicitly respect metric spaces to be able to produce testable recommender systems, retrieving similar items, personalized recommendations, interpretable recommendations, scalable recommendations, and the ability to use recommender embeddings for other tasks such as predicting categorical item side information.
- Develop deep embeddings that are robust towards adversarial attacks.
- Develop embeddings that remove the bias towards sensitive attributes such as race, gender, and age that may be present in the dataset.

## 8.3 Concluding Statement

The thesis has shown deep graph embeddings are a powerful tool for recommender systems that can be adapted to novel metrics of practical commercial use. It is hoped that the thesis provides novel insights into how and why these graph-based methods work and pave the way for future extensions that further explore their potential.

# Bibliography


[1] How retailers can keep up with consumers — mckinsey. https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers. (Accessed on 07/18/2019).

[2] Netflix recommendation engine worth $1 billion per year - business insider. https://www.businessinsider.com/netflix-recommendation-engine-worth-1-billion-per-year-2016-6. (Accessed on 07/18/2019).

[3] A practical guide to building recommender systems from algorithms to product. https://buildingrecommenders.wordpress.com/. (Accessed on 08/05/2019).

[4] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Controlling popularity bias in learning-to-rank recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 42–46. ACM, 2017.

[5] Chris Anderson. *The long tail: Why the future of business is selling less of more*. Hachette Books, 2006.

[6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[7] Saaid Baraty, Dan A Simovici, and Catalin Zara. The impact of triangular inequality violations on medoid-based clustering. In *International Symposium on Methodologies for Intelligent Systems*, pages 280–289. Springer, 2011.

[8] Alejandro Bellogín, Pablo Castells, and Iván Cantador. Statistical biases in information retrieval metrics for recommender systems. *Information Retrieval Journal*, 20(6):606–634, 2017.

[9] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[10] Avishek Joey Bose and William Hamilton. Compositional fairness constraints for graph embeddings. *arXiv preprint arXiv:1905.10674*, 2019.

[11] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[12] Rocío Cañamares and Pablo Castells. Should i follow the crowd?: A probabilistic analysis of the effectiveness of popularity in recommender systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 415–424. ACM, 2018.

[13] Jaime G Carbonell and Jade Goldstein. The use of mmr and diversity-based reranking for reodering documents and producing summaries. 1998.

[14] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In *Machine Learning for Healthcare Conference*, pages 301–318, 2016.

[15] Evangelia Christakopoulou and George Karypis. Local item-item models for top-n recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 67–74. ACM, 2016.

[16] A Christopher, M Andrew, and S Stefan. Locally weighted learning. *Artif Intell Rev*, 11(1-5):11–73, 1997.

[17] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[18] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.

[19] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[20] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[21] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[22] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[23] Krzysztof Dembczynski, Arkadiusz Jachnik, Wojciech Kotlowski, Willem Waegeman, and Eyke Hüllermeier. Optimizing the f-measure in multi-label classification: Plug-in rule approach versus structured loss minimization. In *International conference on machine learning*, pages 1130–1138, 2013.

[24] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 147–153, 2003.

[25] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[26] Sharad Goel, Andrei Broder, Evgeniy Gabrilovich, and Bo Pang. Anatomy of the long tail: ordinary people with extraordinary tastes. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 201–210. ACM, 2010.

[27] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1809–1818. ACM, 2015.

[28] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[29] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[30] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[31] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[33] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[34] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative metric learning. In *Proceedings of the 26th international conference on world wide web*, pages 193–201. International World Wide Web Conferences Steering Committee, 2017.

[35] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee, 2008.

[36] Jaya Kawale, Hung H Bui, Branislav Kveton, Long Tran-Thanh, and Sanjay Chawla. Efficient thompson sampling for online matrix-factorization recommendation. In *Advances in neural information processing systems*, pages 1297–1305, 2015.

[37] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[39] Marzena Kryszkiewicz and Piotr Lasek. Ti-dbscan: Clustering with dbscan by means of the triangle inequality. In *International Conference on Rough Sets and Current Trends in Computing*, pages 60–69. Springer, 2010.

[40] Oleksii Kuchaiev and Boris Ginsburg. Training deep autoencoders for collaborative filtering. *arXiv preprint arXiv:1708.01715*, 2017.

[41] Brian Kulis et al. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013.

[42] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[43] Neal Lathia, Stephen Hailes, Licia Capra, and Xavier Amatriain. Temporal diversity in recommender systems. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 210–217. ACM, 2010.

[44] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[45] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[46] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[47] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[48] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76–80, 2003.

[49] Huan Ling, Jun Gao, Amlan Kar, Wenzheng Chen, and Sanja Fidler. Fast interactive object annotation with curve-gcn. In *CVPR*, 2019.

[50] Qiuxia Lu, Tianqi Chen, Weinan Zhang, Diyi Yang, and Yong Yu. Serendipitous personalized ranking for top-n recommendation. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 258–265. IEEE, 2012.

[51] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in neural information processing systems*, pages 4898–4906, 2016.

[52] Odalric-Ambrym Maillard, Rémi Munos, and Gilles Stoltz. A finite-time analysis of multi-armed bandits problems with kullback-leibler divergences. In *Proceedings of the 24th annual Conference On Learning Theory*, pages 497–514, 2011.

[53] Jérémie Mary, Romaric Gaudel, and Preux Philippe. Bandits warm-up cold recommender systems. *arXiv preprint arXiv:1407.2806*, 2014.

[54] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52. ACM, 2015.

[55] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[56] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[57] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[58] Andrew W Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 397–405. Morgan Kaufmann Publishers Inc., 2000.

[59] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

[60] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 11th IEEE International Conference on Data Mining*, pages 497–506. IEEE, 2011.

[61] Charles C Onu, Jonathan Lebensold, William L Hamilton, and Doina Precup. Neural transfer learning for cry-based diagnosis of perinatal asphyxia. *arXiv preprint arXiv:1906.10199*, 2019.

[62] Yoon-Joo Park and Alexander Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 11–18. ACM, 2008.

[63] Kalyanapuram R Parthasarathy. *Probability measures on metric spaces*, volume 352. American Mathematical Soc., 2005.

[64] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[65] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[66] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[67] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[68] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[69] Suvash Sedhain, Hung Hai Bui, Jaya Kawale, Nikos Vlassis, Branislav Kveton, Aditya Krishna Menon, Trung Bui, and Scott Sanner. Practical linear models for large-scale one-class collaborative filtering. In *IJCAI*, pages 3854–3860, 2016.

[70] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, pages 111–112. ACM, 2015.

[71] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *arXiv preprint arXiv:1211.0053*, 2012.

[72] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[73] Harald Steck. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 125–132. ACM, 2011.

[74] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[75] Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 109–116. ACM, 2011.

[76] Maksims Volkovs, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner. Two-stage model for automatic playlist continuation at scale. In *Proceedings of the ACM Recommender Systems Challenge 2018*, page 9. ACM, 2018.

[77] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. 2018.

[78] Xinxi Wang, Yi Wang, David Hsu, and Ye Wang. Exploration in interactive personalized music recommendation: a reinforcement learning approach. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(1):7, 2014.

[79] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. Learning maximal marginal relevance model via directly optimizing diversity evaluation measures. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 113–122. ACM, 2015.

[80] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017.

[81] Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen Chen. Challenging the long tail recommendation. *Proceedings of the VLDB Endowment*, 5(9):896–907, 2012.

[82] Mi Zhang and Neil Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 123–130. ACM, 2008.

[83] Xiangyu Zhao, Zhendong Niu, and Wei Chen. Opinion-based collaborative filtering to solve popularity bias in recommender systems. In *International Conference on Database and Expert Systems Applications*, pages 426–433. Springer, 2013.

[84] Xiaoxue Zhao, Weinan Zhang, and Jun Wang. Interactive collaborative filtering. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 1411–1420. ACM, 2013.

[85] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S Yu. Spectral collaborative filtering. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 311–319. ACM, 2018.

[86] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM, 2005.