Pure Cold Start Recommendation by Learning on Stochastically Expanded Graphs

Pure Cold Start Recommendation by Learning on Stochastically Expanded Graphs

Thesis

to obtain the degree of Master in Computer Science, Artificial Intelligence Technology track at Delft University of Technology,
to be publicly defended on Thursday, November 24th 2022 at 13:00

by

Simon Ronald Dahrs

Born in Veldhoven, The Netherlands.

Multimedia Computing Group,
Faculty of Electrical Engineering, Mathematics and Computer Science (Faculteit Elektrotechniek, Wiskunde en Informatica),
Delft University of Technology,
Delft, The Netherlands.

Thesis advisor:	Prof.dr.ir. A. Hanjalic, Faculty EEMCS, TU Delft
Daily supervisor:	Dr.ir. E. Isufi, Faculty EEMCS, TU Delft
Daily co-supervisor:	Ir. B. Das, Faculty EEMCS, TU Delft

Keywords: graph neural networks, recommender systems, pure cold start

Summary

Recommender systems help users to find items they presumably like based on data collected on that user. Collaborative filtering is arguably the most common recommendation system technique. It uses collected ratings to compute similarities between users and recommends items based on those similarities. With that, a problem arises when there are few ratings available for a user, also called the cold start problem. An even stricter setting is where no ratings of a user are available at all, called the pure cold start problem. Graphs are a natural way to represent data in this domain and Graph Neural Networks (GNNs) have proven to achieve state-of-the-art accuracies in the domain of recommender systems over the past few years. In this work, we address this problem from a graph perspective. In the current literature, the problem is alleviated by turning to external information, such as demographic or social information. Although these methods work, they cannot be deployed in situations where this information is not available. However, our approach uses only rating data. As ratings are unavailable for pure cold start users, their connectivity to the graph is unknown. To overcome this, we use a stochastic attachment model to infer connectivity. This model is composed of a Bernoulli random variable with the corresponding probability value and edge weight for each existing user. In this work, we propose to learn the parameters of a graph convolution model through empirical risk minimization using the stochastic attachment model to attach users who previously attached to the graph and their ratings, and use it to predict the same for unseen users. Furthermore, we propose to learn the probability and weight values of the stochastic attachment model jointly with the parameters of the graph convolution model. We compare these methods to several graph based and non-graph baselines. Furthermore, we compare different methods such as graph filters, GCNNs, and kNN. The experiments show significant improvements of the graph approaches compared to the non-graph baselines. Furthermore, we show that training on stochastically expanded graphs improves accuracy compared to only testing on them by induction.

ACKNOWLEDGEMENTS

I would like to thank everyone who has helped and supported me throughout the process of this work.

First, I want to thank Elvin and Bishwadeep for supervising my thesis. They guided me through this process not by telling me exactly what to do, but by providing me with the information and resources necessary to develop my own understanding. I started without knowing anything about graph learning, but due to their patience and support, I managed to gain the required knowledge. Furthermore, I want to thank the DelftBlue [13] team for granting me access to the supercomputer on which I executed all the 114 experiments.

I also want to thank my family, especially my brothers, Martijn and Willem, and my parents. If it were not for your love and support throughout my life so far, I would not be where I am today. Thank you, dad, for inspiring me and encouraging me to pursue a career in computer science. Thank you, mom, for always taking care of me.

I want to thank my employer Jaap for supporting me during my academic career. I have always worked with great pleasure and because of your flexibility I was able to combine the job with my studies.

Finally, I want to thank my girlfriend Siri and my friends, in particular Philippe, Marijn, Tim, Julius, Sam, Süleyman, and Tom. All the fun times we had together made this process much more bearable.

Contents

1 INTRODUCTION

As our consumption of multimedia content becomes increasingly digitalized, the abundance of data increases in tandem. Every minute of 2020, 65000 photos were uploaded to Instagram, 500 hours of video were uploaded to YouTube, and 28 songs were added to the Spotify catalog [14, 86]. Instead of actively searching for content using a search engine, a recommender system will passively provide items that are likely relevant to the user so that they can access relevant items with little effort. Not only do the users benefit from recommender systems, but also the commercial enterprises that design them. For example, 35% of the products purchased by customers on Amazon and 75% of what Netflix customers watch, originates from product recommendations [45]. According to Gomez-Uribe and Hunt from Netflix, in 2015 the company saved up to 1B USD per year by offering personalized recommendations [20].

The relevance of an item to a user is determined from the information available about the user and the item. Generally, this information is obtained from historical consumption patterns. For example, in an e-commerce system, we can obtain which items a user previously has bought. The most popular recommendation system technique, collaborative filtering (CF), compares the patterns of the target user with the patterns of other users to identify similar users [59]. From these similar users, we deduce which item our target user will probably like. So, in the e-commerce example, we find users who roughly bought the same items as our target user. Subsequently, we can recommend an item that those similar users bought but our target user did not buy. A drawback of this approach is that we need sufficient rating data from our target user to find similar users. When only a few ratings are available, we cannot accurately identify similar users. This is what is termed the cold start problem [7, 8, 39]. The most extreme case is what is known in the literature as the pure or strict cold start problem [58]. This is the problem when there are no ratings available from a user or item. In commercial systems, this is a frequently occurring scenario. If we again consider the example of an e-commerce system, this would mean that we cannot recommend personalized products on the homepage using CF approaches. Since we have shown that Amazon sales depend significantly on recommended products, this is an expensive problem.

In recent years, Graph Neural Networks (GNNs) have proven to achieve state-of-the-art results in the field of recommender systems [24, 67, 78]. Graph Neural Networks are a collection of neural network architectures that can be applied to irregular data, which can be encoded as graphs. Arguably, the most important component is the graph convolution, which is designed to transfer the proven success of CNNs to the domain of irregular data [34, 40]. We will use the term graph convolution model for any graph model using graph convolutions. These graph convolution models process graphs by using the coupling between its topology and its data, also called the graph signal, to learn patterns end-to-end.

Data in recommender systems can be naturally represented as graphs. Users can be represented by nodes, and two users are connected by an edge if they are similar. In this way, the collaborative nature is incorporated into the graph topology. We can then apply graph signal processing tools, such as GCNN, to perform collaborative filtering. If we now look at the pure cold start problem from a graph perspective, we can reformulate it. As the connectivity between users is determined by rating patterns, which are unknown in this scenario, the connectivity of the pure cold start node to the graph is unknown.

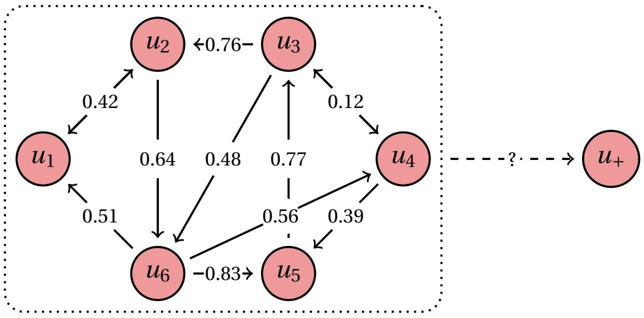


Figure 1.1: Visualization of a pure cold start node in a recommender system graph. The nodes represent the users, and the edge labels are similarities between them. The nodes within the box represent the known users of the system. u_+ represents the pure cold start user for which the connectivity to the graph is unknown. For simplicity, we omitted visualizing the graph signal.

This scenario is visualized in Figure 1.1. Many approaches in the literature solve this problem by using external information to infer user similarity, and thus connectivity, from [26, 50]. Sometimes this data is not available, for example, due to privacy concerns [3, 10, 87].

Therefore, we take it a step further by performing pure cold start recommendation without using external information, but by using stochastic attachment. The current literature already addresses this [12]. However, they use a fixed pretrained graph filter and then learn the stochastic attachment model. Our hypothesis is that we can improve pure cold start recommendation by either (1) using a heuristic stochastic attachment model to attach a pure cold start node and training a graph convolution model on this expanded graph (2) or by jointly optimizing a stochastic attachment model and a graph convolution model. By doing so, the graph convolution model is adapted to the stochastically expanded graphs, potentially yielding a higher accuracy. Specifically, in this thesis, we address the following research questions:

RQ.1 *How can we alleviate the pure cold start problem in collaborative filtering via graph convolutions by training over stochastically expanded graphs?*

RQ.2 *Can we jointly learn a stochastic attachment model and graph convolutions instead of relying on a heuristic attachment model?*

To answer these questions, we compare several graph-based and non-graph methods that can perform pure cold start recommendation. The methods we compare are (1) mean rating (2) SVD++ (3) Erdős-Rényi attachment (4) Barabási-Albert attachment (5) learned attachment. To answer the research questions posed above, we make the following contributions.

1. We use heuristic stochastic methods to attach a pure cold start user to the graph and train a graph convolution model on those stochastically expanded graphs to predict his rating or ranking output.
2. We use a learnable stochastic attachment model to attach a pure cold start user

to the graph and optimize it jointly with a graph convolution model to predict its rating or ranking output by training on those stochastically expanded graphs.

The remainder of this thesis is structured as follows. Chapter 2 provides the theory necessary for this thesis. Chapter 3 covers the relevant literature related to this research. Chapter 4 describes in detail the proposed method. Chapter 5 provides the results and their interpretations. Chapter 6 gives answers to the posed research questions and suggests potential future work.

NOTATION

For the remainder of this thesis, we use the following notations. Matrices are represented by bold uppercase letters (i.e. $\mathbf{A}$), vectors by bold lowercase letters (i.e. $\mathbf{a}$), and scalars by regular weight letters (i.e. a, A). Sets are represented by uppercase calligraphic letters (that is, $\mathscr{A}$). $[\mathbf{A}]_{ij}$ denotes the entry in the ith row and jth column of the matrix $\mathbf{A}$. $[\mathbf{a}]_i$ denotes the ith entry in the vector $\mathbf{a}$.

2 Background

The theory behind recommender systems is expansive due to the many different approaches. As this thesis is concerned with recommender systems implemented by a graph-based collaborative filtering approach, the focus is on these kinds of techniques. Also, the cold start phenomenon is highlighted in this chapter. We provide and explain the required theory that is needed for the rest of this work. Section 2.1.1 introduces the basic theory and terminology of collaborative filtering. In Section 2.1.3 we explain what the cold start is. Then we provide a brief overview of some other relevant techniques used for recommender systems. In Section 2.1.4 we introduce different ways to measure the success of recommender systems. Section 2.2 describes how graphs are present in recommender systems in Section 2.2.1. In Section 2.2.2 we explain what graph convolutions are, and in Section 2.2.3 how they are used in Graph (Convolutional) Neural Networks. Next, in Section 2.2.4, we show how these GNNs are utilized in recommender systems. The chapter is concluded with a summary in Section 2.3.

2.1. RECOMMENDER SYSTEMS

The general problem that a recommender system is concerned with is supplying a user with new relevant items. The problem of recommending items to users includes two subproblems.

The first is rating prediction. Consider a set of users $\mathscr{U}$ with $|\mathscr{U}| = U$ and a set of items $\mathscr{I}$ with $|\mathscr{I}| = I$ where $||$ denotes the set cardinality. The rating prediction problem is defined as the prediction of unknown ratings $r_{u,i}$ for an item $i \in \mathscr{I}$ by a user $u \in \mathscr{U}$. The known (observed) ratings can be placed into a $U \times I$ rating or user-item matrix $\mathbf{R}$ st. $[\mathbf{R}]_{ui} \neq 0$ if user u rated item i and 0 otherwise. Each row vector $u = [r_{u1}, \dots, r_{uI}] \in \mathbb{R}^I$ denotes user u, and each column vector $i = [r_{1i}, \dots, r_{Ui}]^\top \in \mathbb{R}^U$ denotes item i. Therefore, the rating prediction problem can also be seen as a matrix completion problem for the matrix $\mathbf{R}$, as the objective of both is to estimate the missing values (ratings).

The second is top-k recommendation. For recommender systems, not only predicting ratings is important. Users often see a list of recommendations. Simply selecting the items in the top-k highest predicted ratings is suboptimal, as this ignores other aspects of a recommendation system, such as novelty, diversity and serendipity. Therefore, instead of predicting a rating value, top-k recommendation focuses on predicting the top-k most relevant items for a user $u \in \mathscr{U}$, or vice versa, predicting the top-k most relevant users for an item $i \in \mathscr{I}$. Two main types of feedback are the following:

1. Implicit feedback is derived from the interaction between a user and an item. For example, on an e-commerce website, this could be a user visiting a product page or purchasing it. An important characteristic of implicit ratings is that it is difficult to capture negative feedback. As there are often many items and often a user only interacts or buys a small amount of them, there are many unknown ratings. Therefore, a missing rating is ambiguous: either the user dislikes the item or the user simply did not (yet) interact with it, but may like it. There are different ways to treat missing values, for example, by treating all of them as negatives (0s) [27].

2. Explicit feedback is provided consciously by users of a system. For example, many e-commerce platforms such as Amazon feature a 5-star rating system for purchased items. Contrary to implicit unary ratings, this enables the user to also give negative feedback, which benefits the rating prediction.

2.1.1. COLLABORATIVE FILTERING

Collaborative filtering (CF) is the most used technique for recommender systems [1]. The underlying assumption is that similar users have similar tastes or similar items attract similar users. For example, assume that two users u_1 and u_2 have rated items i_1, i_2 and i_3 with a similar high rating, and assume that u_1 rated item i_4 with rating x, while u_2 did not rate i_4. Following the correlation between u_1 and u_2 perceived by their ratings, it makes sense to assume that u_2 will also rate i_4 similarly to u_4. Figure 2.1 shows an example of collaborative filtering. User B and C both interacted with movies 2 and 3. User C also interacted with movie 4. Following the assumption that similar users have similar tastes, we recommend movie 4 to user B. Two main categories of CF are memory-based and model-based.

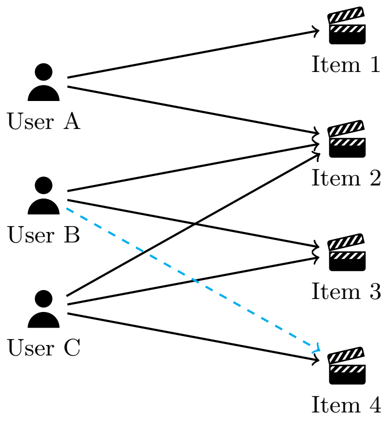


Figure 2.1: Visualization of CF. A solid black arrow denotes observed interaction between the corresponding user and item. The dashed cyan arrow indicates a recommendation.

MEMORY-/NEIGHBORHOOD-BASED

Memory-based CF uses the rating data directly to produce recommendations [8]. The idea is to identify similar (neighboring) items/users to the target user/item according to some similarity measures such as cosine similarity and use that collaborative information to predict a rating for the target. Memory-based CF or neighborhood-based models can be subdivided into two categories: user-user CF and item-item CF. Their approaches are analogous but from a user and item perspective, respectively. For user-user CF, to predict a rating $\hat{r}_{u,i}$ for an item i by user u, similarities between users must be identified using some similarity metric, such as Pearson correlation, defined as

$$corr(u,v) = \frac{\sum_i^I (u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_i^I (u_i - \bar{u})^2}\sqrt{\sum_i^I (v_i - \bar{v})^2}}. \tag{2.1}$$

Then the top-k users most similar to u are collected in the set $\mathcal{N}_u$ and used to predict the rating $\hat{r}_{u,i}$ as

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in \mathcal{N}_u} (r_{v,i} - \bar{r}_v) \cdot corr(u,v)}{\sum_{v \in \mathcal{N}_u} corr(u,v)}. \tag{2.2}$$

Item-item CF is analogous but approaches it from an item perspective. Mostly, an item-based approach is preferred over a user approach if there are more users than items, because in that case, the average item has more feedback than the average user. Therefore, if a new rating is added to the matrix, the average item ratings remain relatively stable compared to the average user rating. If there are more items than users, the opposite is true [38].

MODEL-BASED

In model-based CF, a model is trained on the rating data before making recommendations [59]. After training, inference can be performed on this model to predict missing ratings. Therefore, model-based systems are generally better scalable because they use a model derived from the data instead of directly using the data. Model-based CF methods can roughly be subdivided into two subcategories.

1. The latent factor model decomposes the user-item matrix into smaller matrices. An important latent factor model is matrix factorization. This decomposes the user-item matrix into two matrices $\mathbf{U}$ and $\mathbf{V}$,such that $\mathbf{R} \approx \mathbf{U} \cdot \mathbf{V}^T$. The matrix $\mathbf{U}$ contains d latent factors for each user, while $\mathbf{V}$ contains d latent factors for each item. A predicted rating of an item j by a user i is obtained by taking the dot product of their latent vectors $\mathbf{u}_i^T \cdot \mathbf{v}_j$. By minimizing the loss between the predicted and observed ratings, the model can be trained. Another technique traditionally used to decompose the user-item matrix is SVD [6].

2. Clustering models cluster the user-item matrix before inference is performed [83]. For example, users can be clustered based on their rating similarity. This is cheaper in time complexity compared to memory-based methods to predict an unknown rating to an item by a target user, as only the similarities between the target and the other users in the target user's cluster need to be determined. To reduce the computational burden even more, the user vectors are often reduced using a dimensionality reduction technique. However, compared to matrix factorization models, this approach is still relatively expensive as many similarities need to be computed.

2.1.2. OTHER TECHNIQUES FOR RECOMMENDER SYSTEMS

Besides CF other techniques have been devised for recommender systems.

- Content-based (CB) filtering represents items by their features and only uses the preferences of the target user, also called the user profile [43]. For example, a movie can be represented by its genre, release year, and director. The user profile is represented in the same feature space. Using the interactions or rating history of the target user, the model can be trained with the item representations as input, and the corresponding rating as ground truth. Although CB still needs some preference information from the user, it does not require rating data. Therefore, new or unpopular items will be recommended more frequently compared to CF. Also, for CB, it is possible to construct a user profile by explicitly asking for the preferred feature values.

  Note that classical matrix completion methods use transductive learning, meaning that it reasons from specific training cases, in this case users or items. This causes these methods to not work well for instances unseen during training. For example, the latent factor model has to be retrained for every user or item that is added to update the latent representations for both the existing and new users or items. On the other hand, CB builds a general model and is able to predict ratings for unseen instances, called inductive learning. Therefore, inductive learning is preferred over transductive learning when new instances are introduced frequently.

- Hybrid recommender systems combine different recommender systems to compensate for the drawbacks of each individual system. There exist different approaches to combine recommender systems [9]. For example, a drawback of collaborative filtering systems is the long tail or rich-get-richer problem. This phenomenon occurs because with CF items with few interactions have lower probability of being recommended compared to items with more interactions. To reduce this effect, one can combine the recommendations from CF and a CB by using for example a weighted average, as CB is popularity agnostic. In contrary, a disadvantage of CB is that it will recommend items that are objectively similar to previously interacted items, and therefore lacks serendipity. When the two are combined, CB reduces the richer-get-richer drawback of CF, and CF compensates for the lack of serendipity of CB.

- Another type of recommender system is association rule mining [25]. Relations between items are deduced from co-occurring items in a set of transactions. For example, assume we have a large transaction dataset of some e-commerce system. By analyzing regularities in this set, we obtain sets of items that frequently co-occur, such as {computer, monitor, keyboard}. If a customer then places a computer and a monitor in his shopping cart, the recommender system can recommend a keyboard. As this approach does not rely on user information, this is a form of unpersonalized recommendation.

2.1.3. THE COLD START PROBLEM

The cold start problem in RSs is concerned with uses and items for which the system has no or very little information, which negatively affects the quality of recommendations [39]. This is usually the case for new users and items. For example, a user who just started using an e-commerce website has few or no interactions with items. As CF exploits rating information, it is negatively influenced by this sparsity of user and/or item ratings. The same can hold in the case of item-item CF for items that few or no users rated or interacted with. Note that for CB, only the user's preferences are needed, thus it does not suffer from cold start items, but only from cold start users, given the availability of item content features.

A special case of the cold start problem is the pure cold start problem or the first-time user/item problem. Whereas in the standard case it is assumed that a cold start user or item has only a few interactions, in this special case the user or item strictly has no interactions whatsoever.

There are different ways to deal with the (pure) cold start problem. Common practice is to exploit side information, that is, information about items and users, as explained in more detail in Section 2.1.2 [57].

2.1.4. EVALUATION METRICS

Evaluation metrics measure the efficacy of recommender systems. They can be divided into three categories: (i) User studies (ii) Online evaluation (iii) Offline evaluation [53]. User studies and online evaluations use real people to obtain feedback. In a user study, a user gets a specific task to perform. Online evaluation is concerned with gathering feedback on the system in a real-world setting without informing the user. As often the goal of an RS is to increase interactions or purchases, this is the most direct evaluation metric. Offline evaluation is done with mathematical metrics. They use pre-obtained information from users to calculate scores. The assumption is that this data accurately reflects the current users' behavior or preferences. In this thesis, we will focus on offline evaluation metrics, as these are the most obvious to mutually compare different systems, and do not require interaction with real users. We will consider two classes of metrics, rating prediction metrics and top-k ranking metrics.

Rating prediction metrics
To measure the accuracy of predicted ratings, the mean absolute error (MAE) is often used, which is defined as

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |r_{u,i} - \hat{r}_{u,i}|, \tag{2.3}$$

where $r_{u,i}$ is the ground truth and $\hat{r}_{u,i}$ the predicted rating of an item i by a user u. Metrics related to the MAE are the normalized mean absolute error (NMAE), mean squared error (MSE), and root mean squared error (RMSE). The mean squared error (MSE) places greater emphasis on outliers. To compute the RMSE we take the square root of the MSE,

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (r_{u,i} - \hat{r}_{u,i})^2}. \tag{2.4}$$

Both the MAE and RMSE are represented in the same unit as the prediction variable. However, the RMSE increases when the variance of the error distribution increases, while the MAE represents the mean of the error distribution.

Top-K recommendation
The previous metrics consider the recommendation problem as a rating prediction problem. However, in the real-world, recommendation is in fact a problem of ranking a list of items. To compute this performance, we must know which items in the test set are considered relevant or not. To this end, several approaches can be applied. One is to compute the median rating per user and to consider all items with a rating higher than or equal to be relevant and lower to be irrelevant to him. Another approach is to consider the top-k highest rated items for each user to be relevant to him. The precision of the recommended items is measured as

$$Precision@K = \frac{\#\text{relevant items in top-}K\text{ recommendations}}{K}, \tag{2.5}$$

Often we care about the average precision up to K instead of only a single value of K which is calculated by the average precision (AP) defined as

$$AP@K = \frac{1}{R} \sum_{k=1}^{K} precision@k \cdot rel(k), \tag{2.6}$$

where R is the total number of relevant items for the user, $rel(k) = 1$ if the item in rank k is relevant, otherwise $rel(k) = 0$. The MAP is simply the mean of the APs for all users in the system.

Another important aspect of top-k recommendation is to maximize the coverage of all relevant items, meaning that we should recommend as many different relevant items as possible. This is measured with recall, which is defined as

$$Recall@K = \frac{\#\text{relevant items in top-}K\text{ recommendations}}{\#\text{relevant items for the user}}. \tag{2.7}$$

Note that this gives a more fair comparison between a user with only a few relevant items and a user with many relevant items, as the precision will be lower for the first, but the recall can be more similar.

Whereas recall treats every item in the recommended list equally, the normalized discounted cumulative gain (NDCG) considers an item's position in the list. We start with the discounted cumulative gain (DCG), defined as

$$DCG(N) = \sum_{i}^{N} \frac{g_i}{\log_2(i+1)}. \tag{2.8}$$

which simply sums the relevance scores (gains) for each of the N recommended items, and discounts by the rank of the item. This gain can be defined in different ways, for example, as a binary or ordinal relevance score. The lower the rank of the recommended item, the lesser its relevance contributes to the cumulative gain. As the DCG increases monotonically with the length N of the recommendation list, we cannot compare the DCG for different values of N. To account for this, we can normalize DCG(N) by the ideal DCG(N), which is the optimal ranking up to N items. So, the formula for the NDCG is

$$NDCG(N) = \frac{DCG(N)}{IDCG(N)}. \tag{2.9}$$

2.2. Graph Learning

A graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is a mathematical structure with a set of nodes $\mathcal{V} = \{v_1, \ldots, v_N\}$ and a set of edges $\mathcal{E} = \{(v_i, v_j)\}$ for all pairs of connected nodes (v_i, v_j). There are two types of edges: directed and undirected. An undirected edge has no direction. A directed edge (u, v) is directed from its tail u to its head v. An edge can have a weight, which makes the graph weighted. A graph can be represented by its adjacency matrix $\mathbf{A}$. For an unweighted graph, $\mathbf{A} \in \mathbb{R}^{N \times N}$, with $A_{ij} \neq 0$ if and only if $(i, j) \in \mathcal{E}$. For an undirected graph, its adjacency matrix $\mathbf{A}$ is symmetric, i.e. $\mathbf{A}_{ij} = \mathbf{A}_{ji}$. The number of edges connected to a node is called its degree. For a directed graph, two types of degree can be identified, the in-degree and the out-degree, denoting the number of incoming or outgoing edges, respectively. The degrees of each node can be collected in a degree matrix. If all nodes and edges represent the same entity type or relation type, respectively, the graph is homogeneous; otherwise, it is heterogeneous.

A general algebraic representation of a graph is the graph shift operator (GSO). For a GSO $\mathbf{S}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ we have $\mathcal{S}_{uv} \neq 0$ if $(u, v) \in \mathcal{E}$, otherwise 0. Looking at this definition, we can see that the adjacency matrix is an instance of a graph shift operator. Another instance of GSO is the Laplacian matrix, defined by $L = D - A$, where D is the degree matrix and A the adjacency matrix. Since directed graphs have two different types of degree, one can construct an in-degree Laplacian and an out-degree Laplacian.

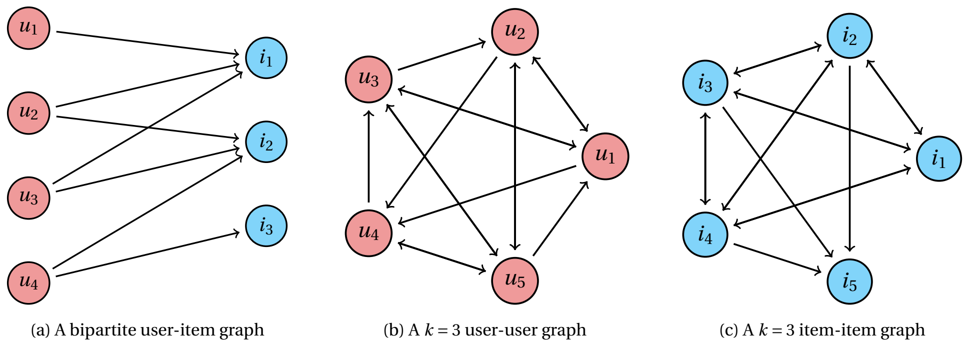

u_1
u_2
u_3
u_4
i_1
i_2
i_3
(a) A bipartite user-item graph
u_1
u_2
u_3
u_4
u_5
(b) A $k = 3$ user-user graph
i_1
i_2
i_3
i_4
i_5
(c) A $k = 3$ item-item graph

Figure 2.2: Different types of graphs in recommender systems. Red nodes represent users, blue nodes represent items.

2.2.1. GRAPHS IN RECOMMENDER SYSTEMS

Graphs are a natural way of representing data in recommender systems. We will consider three different types of graph related to RSs.

1. Rating-based graphs use the rating matrix $\mathbf{R}$ to construct a graph. For example, the user-item matrix $\mathbf{B}$, can be represented as a bipartite graph as shown in Figure 2.2a. A bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a graph whose set of nodes $\mathcal{V}$ can be split into two disjoint sets $\mathcal{U}$ and $\mathcal{I}$, which contain only users and items, respectively. Such a graph is heterogeneous as users and items are different types of entity, and also the relationship types can differ if edges represent different ratings, for example, positive and negative. For a bipartite graph, we have $(u_i, u_j) \notin \mathcal{E}, \forall u_i, u_j \in \mathcal{U}$ and $(i_i, i_j) \notin \mathcal{E}, \forall i_i, i_j \in \mathcal{I}$. The adjacency matrix of such a user-item bipartite graph is of the form
$$A = \begin{bmatrix} [0]_{U\times U} & \mathbf{B} \\ \mathbf{B}^T & [0]_{I\times I} \end{bmatrix}. \tag{2.10}$$
Alternatively, user-user and item-item graphs can also be constructed. Each user can be represented as a vector of his ratings for the items in the system, $\mathbf{u} \in \mathbb{R}^I$. Then all mutual similarities can be calculated between all users to obtain a symmetric similarity matrix $\mathbf{A} \in \mathbb{R}^{U\times U}$. Also, other information, such as user or item attributes, can be considered to compute similarities. The similarity matrix can be considered the adjacency matrix of a user-user graph, which is a fully connected weighted graph with $\mathbf{A}(u, v) = sim(u, v), \forall u, v \in \mathcal{U}$. Often only an edge directing from u to v is preserved if u is in the top-k most similar users of v, making it a k-nearest neighbor graph, as shown in Figure 2.2b. Since the nearest neighbor relation function is asymmetric, the graph is directed.
For an item-item kNN graph, each item is represented by the vector containing the ratings it received from all users of the system, $\mathbf{i} \in \mathbb{R}^U$. In the same way as for the user-user graph, a kNN graph as shown in Figure 2.2c can be constructed by maintaining links to the k most similar items for each item in the graph.

2. Knowledge graphs are another type of graph used in recommender systems [78]. It encodes different types of factual relationships between different types of nodes. A

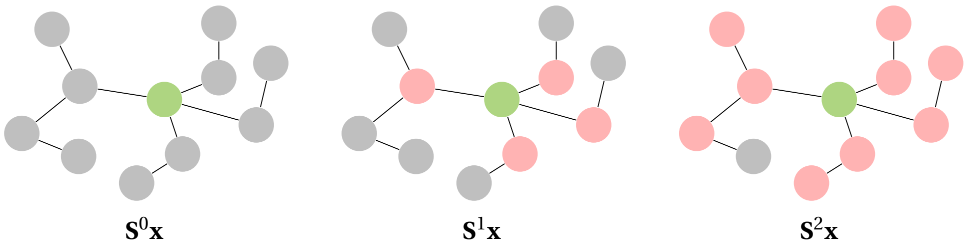


Figure 2.3: The signal of a node diffuses through its neighborhood by applying the GSO recursively. The diffusion is shown from the perspective of the green node. In the leftmost figure no diffusion has taken place. In the middle one, the signal of the green node has been diffused to its direct, or one-hop neighbors. At the rightmost, its signal has been diffused to the two-hop neighborhood.

knowledge graph can, for example, contain an edge between a node representing the movie "Interstellar" and the node representing the genre "Adventure". The label of that edge would be anything similar to "has genre". This type of graph can be used to perform content-based filtering in combination with collaborative filtering to alleviate the cold start problem [50, 72].

3. Social graphs encode different social relations, such as *friend of* or *married to*, between nodes that represent people [60]. This data can be exploited for collaborative filtering. The intuition is that users are often influenced by their social network when it comes to prefering certain items. So, if the social network of a target user is known and some people in that network rated items, these ratings can be used to predict the ratings of the target user. Especially if the target user is a cold starter, this side information can be useful [41].

2.2.2. GRAPH CONVOLUTIONS

The graphs described in Section 2.2.1 expose only a topology. However, each node also has a value assigned to it. In the user-user graph in Figure 2.2b, each node (user) has a scalar value equal to the rating of a certain item by that user. This value can be represented by the function $f : \mathcal{V} \rightarrow \mathbb{R}$. The values at the nodes can be modeled as a signal, which is called the graph signal $\mathbf{x} \in \mathbf{R}^I$. The graph signal can be diffused to neighboring nodes via the edges by performing a graph shift operation. This operation is formulated as

$$\mathbf{x}^{(1)} = \mathbf{S}\mathbf{x}. \tag{2.11}$$

$\mathbf{x}^{(1)}$ is the one-shift of the original graph signal $\mathbf{x}$.
The one-shifted signal in Equation (2.11) diffuses the signals of the nodes to their direct, also called one-hop, neighbors. An important property of the GSO is its locality property. The definition of the one-hop shift at a certain node i is $\mathbf{x}_i^1 = \mathbf{S}^1\mathbf{x}$ and can be transformed into $\sum_{j=1}^{N} S_{ij}x_j = \sum_{j\in\mathcal{N}_i}^{N} S_{ij}x_j$. As this one-hop shift operation only uses the direct neighbors of each node, this operation can be computed locally.

By recursively performing graph shifts, the signal of a node is propagated to nodes a hop further away at each run. Concretely, to diffuse a node's signal to its 3-hop neighbors, the shift operation has to be applied recursively for 3 times. This can be generalized in the recursive definition of a k-hop shift operation, as

$$\mathbf{x}^{(k)} = \mathbf{S}^k\mathbf{x} = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x}) = \mathbf{S}\mathbf{x}^{k-1}. \tag{2.12}$$

Using the definition of Equation (2.12), we can define convolution for graph signals as a graph filter written as polynomial of GSO $\mathbf{S}$:

$$\mathbf{y} = \mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K} h_k\mathbf{S}^k\mathbf{x}. \tag{2.13}$$

The parameter k defines the filter order, which is the number of hops of the operation, similar to the kernel size in a convolutional layer of a CNN. $\mathbf{H}(\mathbf{S})$ is the Finite Impulse Response (FIR) graph filter, with $\mathbf{h}^\top \in \mathbf{R}^{K+1}$ as its vector of coefficients.

2.2.3. GRAPH NEURAL NETWORKS

Graph Neural Networks (GNNs) leverage the graph topology to learn nonlinear representations of the graph and its signal [80]. An important type of GNN is the Graph Convolutional Neural Network (GCNN).

GCNNs consist of L layers containing graph convolutional filters followed by a nonlinear function σ. Each layer l can have a parallel bank of graph filters, producing F_l graph signals or features $\{\mathbf{x}_l^f\}_{f=1}^{F_l}$. Subsequently, each filter takes as input F_{l_1} features, $\{\mathbf{x}_{l-1}^g\}_{g=1}^{F_{l-1}}$. If $l = 1$, the input is the input graph signal $\mathbf{x}$. Each graph filter $f \in 1,\ldots,F_l$ in a GCNN layer l has a vector of coefficients $\mathbf{h}_l^{fg} = \{h_{0l}^{fg},\ldots,h_{Kl}^{fg}\}$ for each F_{l-1} input feature g. The coefficients of a single graph filter for all input features can be collected in matrix $\mathbf{H}_l^f = [\mathbf{h}_l^{f0},\ldots,\mathbf{h}_l^{fF_{l-1}}]$. From this definition we can derive that a layer in a GCNN has $(K+1)\cdot(F_l)\cdot(F_{l-1})$ parameters. With these parameters, we can compute a convolved feature f using a multilayer GCNN as

$$\mathbf{z}_l^f = \sum_{g=1}^{F_{l-1}}\sum_{k=0}^{K} h_{kl}^{fg}\mathbf{S}^k\mathbf{x}_{l-1}^g = \sum_{g=1}^{F_{l-1}} \mathbf{h}_l^{fg}(\mathbf{S})\mathbf{x}_{l-1}^g, \text{for } f = 1,\ldots,F_l. \tag{2.14}$$

The convolution filter is used to linearly combine neighboring data points for the current target node. The filter uses the same shared weights in the linear combination for every target node. This preserves translation-equivariance, which is a welcome feature. To acquire the final output feature $\mathbf{x}_l^f$, we feed $\mathbf{z}_l^f$ into an activation function σ,

$$\mathbf{x}_l^f = \sigma(\mathbf{z}_l^f) \tag{2.15}$$

[15]. Figure 2.4 visualizes GCNN processing a graph signal. The output features of the final convolutional layer L can, for example, be fed into a fully connected layer to compute the final output of the GCNN, for example a graph class label.

2.2.4. GRAPH CONVOLUTIONS AND NEURAL NETWORKS IN RECOMMENDER SYSTEMS

We can structure the user-item rating data as graphs, as described in Section 2.2.1. For example, a user-user graph connects users to their k most similar users computed from

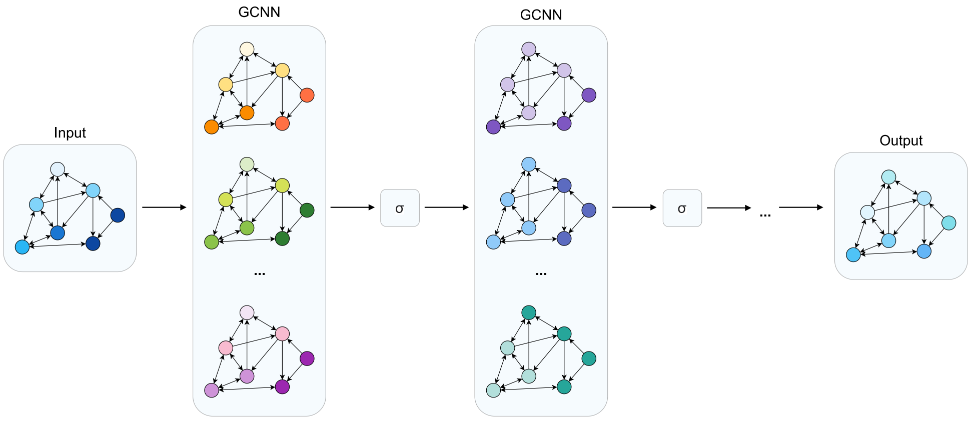


Figure 2.4: Visualization of a GCNN processing a graph and corresponding signal. A *GCNN* block represents a parallel bank of graph filters. The inside of the block displays the current state of the data. Different node colors mean different graph signal values. σ denotes a nonlinearity.

their ratings. We can compute the similarities between users as $[\mathbf{B}]_{u,v} = sim([\mathbf{R}]_u, [\mathbf{R}]_v)$, to obtain the similarity matrix $\mathbf{B} \in \mathbb{R}^{U \times U}$ from the available rating matrix $\mathbf{R}$ and some similarity measure sim such as Pearson's correlation. We then define the graph shift operator of the kNN CF graph for an item i as

$$[\mathbf{G}_i]_{uv} = \begin{cases} \mathbf{B}_{uv}, & \text{if } v \in \mathcal{K}_{u,i} \\ 0, & \text{otherwise} \end{cases}, \tag{2.16}$$

where $\mathcal{K}_{u,i}$ is the set of k users most similar to u who rated the item i. By this definition, the graph contains users that rated item i and every user is connected to its k most similar users in that graph. The signal $\mathbf{x}_i = [\mathbf{R}]_{:,i}$ comprises the ratings of i. By applying a graph convolution filter on this graph and signal, the ratings of i are passed to similar users. Each user receives his own ratings and the ratings of his K-hop neighborhood. To predict a rating $\hat{r}_{ui}$, we can use a graph filter which computes a weighted combination of the ratings of i by the k-hop neighborhood of u, via the equation

$$\hat{r}_{ui} = [\hat{\mathbf{x}}_i]_u = \sum_{k=0}^{K} h_k \mathbf{G}_i^k \mathbf{x}_i = [h_0 \cdot \mathbf{x}_i + h_1 \cdot \mathbf{G}_i \mathbf{x}_i + h_2 \cdot \mathbf{G}_i^2 \mathbf{x}_i + \ldots + h_K \cdot \mathbf{G}_i^K \mathbf{x}_i]_u. \tag{2.17}$$

We collect the shifted signals in $\mathbf{G}_{x_i} = [\mathbf{x}_i, \mathbf{G}_i \mathbf{x}_i, \mathbf{G}_i^2 \mathbf{x}_i, \ldots, \mathbf{G}_i^K \mathbf{x}_i]$, such that $\hat{\mathbf{x}}_i = \mathbf{G}_{x_i} \mathbf{h}$. Let $\mathcal{O}$ be the set of all observed ratings, that is, $\mathcal{O} = \{(u,i) | r_{ui} \neq 0\}$ where $r_{ui} = [\mathbf{R}]_{ui}$ is the true rating value item that user u provided to item i. To train the parameters of the graph filter, we must construct a training set $\mathcal{T} \subset \mathcal{O}$ and define some loss function $\mathcal{L}$. The problem is then to minimize the loss function by changing the graph filter parameters $\mathbf{h}$, formulated as

$$\mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} = \sum_{(u,i) \in \mathcal{T}} \mathcal{L}([\mathbf{G}_{x_i} \mathbf{h}]_u, r_{ui}) + \alpha \|\mathbf{h}\|_2^2, \tag{2.18}$$

where $\|\mathbf{h}\|_2^2$ is the L2 regularization term tuned by α to prevent the coefficients from growing too large. Instead of a graph filter, we can also use a GCNN $f(\mathbf{x}; \mathbf{S}; \Theta)$ on the user-user graph, with Θ denoting all parameters of the GCNN, $h_{l0}^{fg}, \dots, h_{lK}^{fg}$ for each layer $l \in 1, \dots, L$. The general optimization problem is then formulated as

$$\mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} = \sum_{(u,i)\in\mathcal{T}} \mathcal{L}(r_{ui}, [f(\mathbf{x}_i;\mathbf{G}_i;\Theta)]_u) + \alpha\|\Theta\|_2^2 \tag{2.19}$$

where $\|\Theta\|_2^2$ is the L2 regularization term tuned by α. If we instantiate $\mathcal{L}$ as the MSE, the loss L is computed as

$$L = \frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (r_{ui} - [f(\mathbf{x}_i;\mathbf{G}_i;\Theta)]_u)^2 + \alpha\|\Theta\|_2^2. \tag{2.20}$$

We can use stochastic gradient descent (SGD) with backpropagation to minimize $\mathcal{L}$ with respect to Θ [2]. As stated before, for recommender systems, not only rating prediction is important, but also ranking prediction. To optimize for ranking performance, we can use the BPR loss, focusing on relative user preference instead of absolute ratings. To train with BPR loss, we must construct a different training set containing triplets $\mathcal{T} = \{(u,i,j)|r_{ui} >_u r_{uj}\}$. Using $\mathcal{T}$, we can define the likelihood function as

$$\mathbb{P}(r_{ui} >_u r_{uj}|\mathbf{h}) = \sigma(\hat{r}_{ui} - \hat{r}_{uj}) = \sigma([\hat{\mathbf{x}}_i]_u - [\hat{\mathbf{x}}_j]_u), \tag{2.21}$$

where σ is the sigmoid function and $[\hat{\mathbf{x}}_i]_u$ and $[\hat{\mathbf{x}}_j]_u$ can be acquired from a graph filter or a GCNN as defined above. The training set is then defined as $\mathcal{T} = \{(u,i,j)|(u,i) \in \mathcal{O}, (u,j) \notin \mathcal{O}\}$. With this we can formulate the L2 regularized BPR loss as

$$\mathcal{L} = -\sum_{(u,i,j)\in\mathcal{T}} \ln\sigma([\hat{\mathbf{x}}_i]_u - [\hat{\mathbf{x}}_j]_u) + \alpha\|\Theta\|_2^2 \tag{2.22}$$

which we again train using SGD with backpropagation to acquire optimal values for Θ that maximize the log likelihood.

2.2.5. Cold Start in Graph-Based Recommendation

As with other techniques, graph-based recommender systems suffer from the cold start problem. When the ratings are sparse, the similarities in **B** are based on only a few interactions. If we consider user graphs, the neighborhood of a cold target user is inaccurate as it is based on a few interactions. Since the predicted rating for a target user is based on his neighborhood, the accuracy suffers from the sparse ratings.

In the pure cold start scenario, no interactions are available for the target user. Similar users cannot be identified as there are no ratings to use for it. Therefore, the connectivity of a pure cold start user is unknown and accordingly, we cannot predict his rating using his neighborhood. To alleviate this problem, one can turn to other sources of connectivity, for example, attribute similarity.

2.3. Conclusion

In this chapter, we introduced the background information needed for the rest of this thesis. First, we introduced the basics of recommender systems and collaborative filtering (CF) in particular. We also introduced the cold start problem and pure cold start problems which will play a major role in this thesis. The evaluation metrics mentioned will be used to compare the results in this thesis with existing baselines. Next, we explained how the underlying data in recommender systems can be structered as a graph. Finally, we explained graph convolutions and how they are used and optimized in graph neural networks.

3 Literature Review

This chapter provides an overview of the literature related to this work. It is divided into three sections, which will descend from more generally to more specifically related papers. Section 3.1 discusses existing approaches to utilizing GNNs for collaborative filtering, including their general architecture. Subsequently, Section 3.2 provides an overview of how current works address the cold start problem with Graph Neural Networks. The final Section 3.3 covers multiple researches that apply GNNs to learn to perform tasks that involve dynamic graphs.

3.1. Graph Neural Networks for Collaborative Filtering

We will discuss here collaborative filtering approaches implemented with GCNNs and Graph Attention Networks (GATS), respectively, as these are the most prevalent GNN architectures for this use.

GCNNs are arguably the most common type of GNN used for collaborative filtering. Collaborative filtering using deep learning on graphs was first introduced in [47], in which the authors proposed combining a GCN with an RNN to solve the matrix completion problem. Their approach builds user-user and item-item kNN graphs and feeds them into a GCN to extract features. These graphs can be constructed directly from the original rating matrix, but also from content features. These features are subsequently fed into an RNN architecture. This outputs a small change to the rating matrix as side information, and the rating matrix is updated accordingly, so the objective function is a combination of GCN and MF. Instead, the GC-MC model proposed by Berg et al. [5] works directly on the bipartite rating matrix. It uses a single GCN layer, thus only considering the one-hop neighborhood of nodes. GC-MC is uncapable of performing inductive tasks as it uses one-hot node representations, and therefore new nodes cannot be represented. This architecture supports the integration of side information, such as item content features. STAR-GCN overcomes the scalability issue of GC-MC by directly learning user and item embeddings [88], enabling users unrepresented in the training set to be represented during testing. To learn these embeddings, it exploits content information and the graph's topological information. By masking parts of the ground truth user and item embeddings, STAR-GCN learns to generalize to unseen nodes. PinSage focuses on scaling GCNs to industrial settings [85]. Instead of using the full graph and feature matrices as in previous works, PinSage samples the neighborhood to perform more efficient localized computations. SpectralCF performs collaborative filtering in the frequency domain [89]. It stacks spectral convolutional layers and applies them to the bipartite rating graph to learn the collaborative signal. In contract with PinSage, SpectralCF has high computational complexity, as it computes the eigen-decomposition of the graph.

An important work that only uses a GCNN for collaborative filtering is NGCF [67]. It propagates the node embeddings to the K-hop neighborhood in the bipartite rating graph. It concatenates the resulting $K+1$ node embeddings and uses the dot product to predict ratings. Chen et al. argue that the nonlinear feature transformation hurts the collaborative filtering performance [11]. Therefore, they propose LR-GCCF which is a linear model. Different from NGCF, it discards the feature nonlinear transformation, as the authors show that it benefits the recommendations' performance. The authors argue, inspired by ResNets [23], that using the final embeddings alone will lead to worse performance due to the oversmoothing problem. LR-GCCF uses residual rating prediction, which means that the ratings are computed using the embeddings at each layer. The authors prove that this is equivalent to concatenating the embeddings at each layer and using this concatenation to predict the ratings. In fact, this concatenation of $K+1$ embeddings in each layer is exactly the same as done by NGCF [67]. He et al. take it one step further by showing that not only the nonlinearity in a GCNN is redundant or even harmful for the performance of CF, but also the entire feature transformation matrix is [24].

They adopt the NGCF model and trim it down to the essential component by conducting an ablation study, and call their resulting model LightGCN. LightGCN consists solely of the neighborhood aggregation step, thus eliminating the feature transformation and nonlinearity. They argue and empirically prove that, as collaborative filtering only has the ID feature of no semantic meaning, the feature transformation and subsequent nonlinear function of the embeddings harm the prediction performance, as it complicates training. Also, instead of concatenating, they sum $K+1$ embeddings, as the authors argue that the principle of GCNNs is to improve embeddings by propagation.

As many rating datasets consist of implicit feedback, ratings alone do not reflect the different types of interaction, such as clicking, buying, or sharing an item. Those different user intents provide useful information for a recommender systems, and improve its explainability. To this end, people introduced GNN architectures that are capable of modeling different types of intent. Multi-Behavior Graph Convolutional Network (MBGCN) constructs a single rating graph using multi-behavior data [33]. For each type of intent (behavior), there is a corresponding edge type connecting users to items. By applying a GCNN to this graph, we obtain the probability that a user will interact with an item with respect to a specific target intent. Instead of using multi-behavior data as input, which is not always available, Wang et al. propose Disentangled Graph Collaborative Filtering (DGCF) [68]. DGCF constructs K intent-aware graphs, one for each latent intent, and learns intent-specific user and item embeddings for each of them. To enforce independence between the intents to avoid redundancy, DGCF uses a distance correlation between the embeddings. The authors show that LightGCN is a specialization of DGCF with a single intent. Another work that uses more than one graph is [30]. The authors argue that previous graph-based recommender systems focus exclusively on achieving high accuracy, and diversity is not addressed while it is important. Therefore, they construct nearest neighbor and furthest neighbor graphs to achieve high accuracy and diversity, respectively. By jointly optimizing GCNNs over these graphs, they achieve high diversity gains traded for only small accuracy decreases.

The aforementioned works all approach the recommendation problem as a supervised learning problem; they use the observed ratings as ground truth signal. Inspired by recent successes of self-supervised learning in other fields such as computer vision [31, 36, 42], Wu et al. propose a self-supervised graph learning (SGL) framework for recommendation [75]. Using edge dropout, node dropout, and random walk, they stochastically generate different subgraphs, which they call views, for the same node. By using a contrastive loss function, they encourage the similarity of the embeddings of the positive pairs in the two different views to be high and the similarity of the negative pairs to be low. The contrastive loss is combined with a standard BPR loss, and the two tasks can be optimized jointly. SGL is model-agnostic, and the authors implement it on LightGCN. It shows significant improvements, especially for the long-tail problem, referring to the degenerating performance on low-degree nodes, also known as cold-start nodes.

Instead of using only a GCNN, many works propose using a GAT in combination with a GCNN to discriminate between neighbors. The work in [70] and [71] both propose GAT models to disentangle the rating signal into latent representations of user motivations, similar to DGCF [68]. However, [70] and [71] use attention on content features to disentangle the rating signal, while DGCF only uses the rating signal and convolution operations. The authors argue that people have different types of motivation to purchase items, such as its appearance and price. Instead of computing attention scores using neighboring node features, they can also be computed from edge features [48, 82]. Doing so increases the expressive power of a GAT as it can distinguish connections to nodes with identical node features. Edge features can be derived from data, or aggregated from the features of nodes it connects.

3.2. COLD START

Similarly to other types of RS, GNN-based RSs suffer from the cold start problem. A more strict definition of it is the pure cold start problem, which means that a user or item has no associated ratings whatsoever. There are several works on finding means to alleviate the general and pure cold start problem for GNN-based RSs. There are two main principles to alleviate the cold start problem. One is using side information and the other is using the graph topology.

The use of external information, such as social or knowledge graphs, can compensate for the sparsity of the rating data. Combining interaction data with side information enables the generation of recommendations for users or items with sparse ratings compared to exclusively using interaction data.

A common approach to using side information for better cold start recommendations is using attribute data. Users, for example, can have attributes such as age, gender, and country, items can have attributes such as product category, price and color. HERS is an example of a GNN architecture that uses social relationships of users and attribute attributes of items to make recommendations [26]. It builds a user-user graph based on social relations and a kNN item-item graph based on attribute similarities. By aggregating neighbors, it exploits the influence of neighboring items or users to generate recommendations for cold start users and items. Another work is [50] in which AGNN is proposed which uses an extended variational auto-encoder (eVAE) to approximate preferences from user and item attributes. This is a VAE extended by a GNN architecture that learns a mapping from user and item attributes to their corresponding ratings. Therefore, to generate recommendations, only a user's or item's attributes are required, making it applicable to the pure cold start scenario.

Another source of side information is a knowledge graph [51, 56, 63–66, 72]. The bipartite rating graph can be coupled with a knowledge graph such that users and/or items are connected by edges of various types with entities in the knowledge graph. Note that this is different from using attribute data, as a knowledge graph encodes relationships between entities, whereas node attributes are not connected. For example, an item in the rating graph can be connected to its corresponding movie entity in the knowledge graph, which in turn is connected to genre entity "action" with an edge or relation "genre". RippleNet-agg aggregates neighboring node representations in a knowledge graph and combines it with their own representation to capture topological information. The final representation of an entity, influenced by its k-hop neighborhood, is fed into a function together with a user representation to compute a rating. When propagating representations through the knowledge graph, RippleNet-agg neglects the difference in importance between relation types for a user. Wang et al. address this issue by learning personalized user-relation scores, which in fact is the same as employing an attention mechanism to the relations (edges) in the aggregation step [65]. Whereas Ripplenet and KGCN use only the knowledge graph, KGAT combines the rating and knowledge graphs into a collaborative knowledge graph (CKG) [66]. Similarly to KGCN, a knowledge-aware attention function is applied before aggregating neighbors. Although KGCN uses a knowledge graph, it fails to generalize for new users as the CKG needs to be reconstructed and the model retrained for every new user because entities and users are of the same node type. To overcome this, Wang et al. propose CKAN which performs collaborative propagation for the rating relations, and knowledge graph propagation separately and combines them afterwards. Previously discussed works [68, 70, 71] argued that different relations between users are represented by the same edges. This is also incorporated by [69] in their model CKIN, which explicitly models users' intents when rating an item [69]. These intents are learned by applying an attention mechanism to the relation embeddings. This improves the recommendation performance, and it also improves the explainability of a recommendation, as it is based on predicted intents. CKIN uses different aggregation schemes for the intent graph and the knowledge graph to exploit behavioral patterns and item-relatedness, respectively. Note that knowledge graph-based recommendation not only increases prediction accuracy and diversity, but also enhances explainability, as factual information is used to generate recommendations.

Instead of augmenting the user-item graph with information on items like knowledge graphs, also information about users' social network can be exploited. GraphRec is a GNN model that uses a user-item graph and a user-user social graph, and learns user embeddings separately from the rating and social graphs [17]. By applying attention, the differences in influence or relevance of users or items for a certain target user are modeled. However, GraphRec only considers the first-order neighbors, leaving the higher-order influences not utilized. Instead, DiffNet models the high-order social influence by recursively ropagating or diffusing user embeddings through the social graph [76]. The improved DiffNet++ adds an interest diffusion layer that captures the collaborative filtering signal [77]. It is very similar to GraphRec but is capable of modeling the higher-order neighborhood.

In many real-world scenarios, however, also side information is sparse or entirely unavailable. Therefore, there is a need for techniques that alleviate the pure cold start problem without requiring side information.

They previously discussed STAR-GCN is a model that handles cold-start nodes by setting some node embeddings to zero entirely to simulate cold-start nodes during training [88]. Subsequently, the model is trained to reconstruct their true embeddings based on the graph topology. Therefore, it does not rely on content information for cold-start users. A similar work is [21] which proposes a pretraining framework to alleviate the cold-start problem using only the graph topology. It also deals with the problem of a low-quality embedding of a cold start node that is propagated to a normal node, harming the embedding of the latter. Another method of applying pretraining focused on cold start recommendation is DropoutNet [62]. Although it performs standard CF instead of graph-based CF, it is still worth examining. Similarly to an auto-encoder, it learns to reconstruct the input rating matrix using a transformed rating matrix and user and item features. It applies dropout to the rating matrix, meaning that some rows or columns (user or item interactions) are set to zero. By doing that, the model learns to recon-

Name	Year	Pure cold start	Uses side information
GC-MC [5]	2017	☐	☑
SpectralCF [89]	2018	☐	☐
DiffNet [76]	2019	☐	☑
GraphRec [17]	2019	☐	☑
HERS [26]	2019	☐	☑
KGAT [66]	2019	☐	☐
KGCN [65]	2019	☐	☑
KGNN-LS [64]	2019	☐	☑
KNI [51]	2019	☑	☑
RippleNet [63]	2019	☐	☑
STAR-GCN [88]	2019	☐	☑
AGNN [50]	2020	☑	☑
DiffNet++ [77]	2020	☐	☑
MBGCN [33]	2020	☐	☑
MetaHIN [44]	2020	☑	☑
MetaCF [73]	2020	☐	☐
HAKG [56]	2021	☐	☑
KGIN [69]	2021	☐	☑
Pre-Train [21]	2021	☐	☐

Table 3.1: GNN-based recommender systems that focus on alleviating the cold start problem.

struct the ratings from the features. For a cold-start user, the model is able to exploit its attributes to generate recommendations. MetaCF is a metalearning framework that focuses on fast adaptation to the few interactions of a cold start user or an item [73]. It can be applied to any differentiable CF method. As MetaCF performs meta-learning, it consists of the actual CF model and the meta-model. The CF model is optimized as normal, and the meta-model learns parameters that enable it to quickly adapt to only a few interactions. The meta-model is then fine-tuned by retraining with the few interactions of a cold start user, resulting in a user-specific model. Another pretraining approach is suggested in [21], which consists of three components. The basic pretraining component only considers the first-order neighbors to predict an embedding for a target node, without considering its own embedding. By doing so, the model learns to quickly adapt to cold start users and items. The authors argue that despite this pretraining deals with cold start target nodes, cold start neighbors affect target nodes in a delayed manner. Therefore, prior to the basic pretraining, the authors propose to use a meta-aggregator. This meta-aggregator applies attention to the neighbors of each neighbor of a target u, and uses those attention scores to aggregate their embeddings. This improves the embedding of the few neighbors of the cold start node u as they differentiate between their neighbors. The final component they propose is the adaptive neighbor sampler, which learns to select the most important neighbors, apart from the first-order neighbors which are all selected.

We organize the above-mentioned literature in Table 3.1. All of them attempt to alleviate the cold-start problem either by incorporating side information or by exploiting the few interactions of a cold-start user or item. These approaches require the availability of content data or some interactions, respectively. Only the approach in [12] does not use side information or ratings of cold start users. However, in a pure cold start scenario, none of these approaches is adequate.

3.3. EXPANDING GRAPH LEARNING

In graph-based collaborative filtering, the pure cold start problem can be framed as the problem of not knowing the attachment of an incoming node. By learning how an incoming node attaches to an existing graph and applying a GNN on the expanded graph, we can predict ratings for pure cold starters.

Dynamic graphs have been researched in various domains, such as computer vision, sensor data analysis, in addition to recommender systems [79, 84].
The work in [32] addresses the case of a graph with incoming nodes with unknown graph signals, also called labels, added to the graph every timestep. Their model learns to predict the label of a new node and receives the true label after some delay. It updates the learned model using this true label. The authors of [54] propose a Temporal Graph Network (TGN) framework which is capable of predicting how a graph evolves by learning from node labels. EvolveGCN employs a Recurrent Neural Network (RNN) in combination with a GCN in order to capture the dynamism of the graph sequence. Whereas other methods apply an RNN to the node embeddings returned by a GCN [46, 49, 55] to learn a node's temporal evolution, EvolveGCN applies it to the parameters of the GCN, and the GCN is used to predict the node embeddings in a future timestep. Instead of using deterministic attachment, Das and Isufi propose using stochastic attachment that can be used for cold start recommendation [12]. They use a fixed graph filter and learn the parameters of the stochastic model by minimizing the MSE of the ratings.

Contrary to the above-mentioned works that use a GCNN as a GNN, [81] uses a special form of GAT [61], coined the Temporal Graph Attention layer (TGAT). This can be employed for dynamic graphs. The authors assume that the meaningfulness of neighboring nodes to a target node is negatively correlated with the moment of connection. TGAT uses some time encoding which is concatenated to the node embedding, and the total is fed into a self-attention mechanism that learns to capture the temporal aspect of the graph.

3.4. CONCLUSION

In this chapter we covered the relevant literature related to this thesis. In Section 3.1, we discussed papers proposing to use GNN architectures to perform collaborative filtering. We covered the discovery that for recommender systems, the (nonlinear) feature transformation harms the performance. Section 3.2 contains papers that perform collaborative filtering using GNNs and focus on the cold start problem. We showed that none of them can do pure cold start recommendation without the use of side information. The proposed methods either turn to side information such as knowledge or social graphs, or they require a few ratings, and are therefore unapplicable to the pure cold start scenario. Finally in Section 3.3 we covered literature on expanding graph learning. Although these methods are not directly applied to collaborative filtering or recommender systems, they are related to our proposed method. Although they can learn how a graph grows, they use node information and do not use stochastic attachment, except for [12]. However, their method optimizes the stochastic model while keeping the graph filter fixed. This is different from our proposals as we use a heuristic stochastic attachment model and train a graph convolution model on the expanded graphs, and we jointly optimize the stochastic model with the graph convolution model. This will adapt the graph convolution model to the task of recommendation for stochastically attached nodes.

4 Stochastic attachment for pure cold start recommendation

This chapter will formulate the problem and provide the contributions of this thesis. The core contribution is to train a graph convolution model on stochastically expanded graphs to do pure cold start recommendation. By doing so, a node with no available information can be attached to an existing graph. The graph topology and available ratings are then exploited by a graph convolution model to predict ratings for the pure cold start node. In addition to heuristic-based stochastic models, we also propose to jointly train a stochastic model and a graph convolution model to optimize recommendation performance. First, in Section 4.1 we describe the stochastic attachment of an incoming node in a general context. Next, in Section 4.2 we explain how we construct nearest-neighbor graphs for collaborative filtering. After that, in Section 4.3 we show how a graph convolution model interpolates the graph signal at a stochastically attached node. Then, in Section 4.4, we state the problems of optimizing the graph convolution coefficients for rating and ranking prediction of pure cold start users. Subsequently, in Section 4.5, we state the problem of jointly optimizing the graph convolution coefficients and the parameters of the stochastic attachment model for these tasks. The different non-graph and graph-based methods that we compare are explained in detail in Section 4.6. Then, Section 4.7 explains how we optimize the parameters of these methods. We conclude the chapter in Section 4.8 with a conclusion.

4.1. STOCHASTIC ATTACHMENT

We consider a directed graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ of N nodes in set $\mathscr{V} = \{v_1, \ldots, v_N\}$ and E edges in set $\mathscr{E} \subseteq \mathscr{V} \times \mathscr{V}$. The adjacency matrix $\mathbf{A}$ is a description of the graph $\mathscr{G}$ for which $A_{i,j} > 0$ if $(i, j) \in \mathscr{E}$. For an incoming node v_+ with unknown connectivity, we can still attach it using stochastic attachment, sampled from a stochastic model. This forms the expanded graph $\mathscr{G}_+ = (\mathscr{V}_+, \mathscr{E}_+)$ with $\mathscr{V}_+ = \mathscr{V} \cup v_+$ and $\mathscr{E}_+ = \mathscr{E} \cup (v_i, v_+)$ for the newly formed edges. The attachment to the graph of node v_+ is characterized by the vector $\mathbf{a}_+ \in \mathbb{R}^N$. The values of this attachment vector are defined as $[\mathbf{a}_+]_i = w_i$ if the node v_i attaches to v_+ with edge weight w_i, and 0 otherwise. The adjacency matrix of this expanded graph is

$$\mathbf{A}_+ = \begin{bmatrix} \mathbf{A} & \mathbf{a}_+ \\ \mathbf{0}^T & 0 \end{bmatrix}, \tag{4.1}$$

where $\mathbf{0}$ is a vector containing N zeros. Note that since this adjacency matrix is asymmetric, the graph is directed. We want the existing nodes to connect to the incoming node, but not vice versa. Therefore, the last column contains the connectivity of all nodes $v_i \in \mathscr{V}$ with v_+, and the last row, which contains the edge weights of edges going out of v_+, is set to all zeros.

We consider a node $v_i \in \mathscr{V}$ to connect to v_+ with probability p_i, forming an edge of weight w_i. Figure 4.1 visualizes this stochastic attachment of v_+. Moreover, we assume that all edges are formed independently. Therefore, the attachment vector is a collection of N weighted Bernoulli random variables. Each entry is now defined as

$$[\mathbf{a}_+]_i = \begin{cases} w_i & \text{with probability } p_i \\ 0 & \text{with probability } 1 - p_i \end{cases}, \tag{4.2}$$

for $i = 1 \ldots N$. We can split each entry $[\mathbf{a}_+]_i$ into a standard Bernoulli distribution, that is, with outcomes w_i and 0 and probability p_i. We collect all probabilities p_i and weights w_i in the vectors $\mathbf{p}$ and $\mathbf{w}$, respectively. $\mathbf{p}$ and $\mathbf{w}$ characterize the stochastic attachment of any node incoming to $\mathscr{G}$. As $\mathbf{a}_+$ is a vector that contains random variables, it has an expectation, which is equal to $\mathbb{E}[\mathbf{a}_+] = \mathbf{p} \circ \mathbf{w}$, with $\circ$ being the Hadamard product. The expected topology of $\mathscr{G}_+$ is described by the expected adjacency matrix $\mathbf{A}_+$,

$$\mathbb{E}[\mathbf{A}_+] = \begin{bmatrix} \mathbf{A} & \mathbf{p} \circ \mathbf{w} \\ \mathbf{0}^T & 0 \end{bmatrix}. \tag{4.3}$$

In practice, we sample the Bernoulli distribution with the parameter p_i for each $i = 1 \ldots N$ to obtain the binary vector $\mathscr{S}(\mathbf{p})$. We then obtain a realization of the attachment vector as $\mathbf{a}_+ = \mathscr{S}(\mathbf{p}) \circ \mathbf{w}$.

Let $\mathbf{x} = [x_1, \ldots, x_N]^\top$ be the graph signal where $x_i \in \mathbf{x}$ maps the node v_i in the existing graph to a value. Graph signal processing techniques process the signal by incorporating its coupling with the topology. We use Graph Filters and Graph Convolutional Neural Networks as tools to perform this processing.

4.2. COLLABORATIVE GRAPH FILTERING

In order to perform Nearest-Neighbor (NN) collaborative filtering, we need graphs that capture the collaborative signal. To do so, we make some modifications to the graph

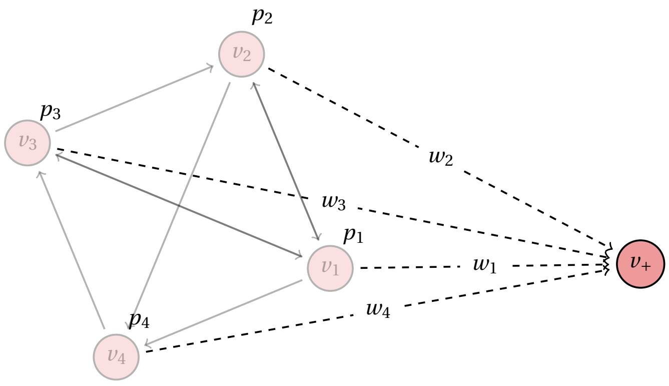


Figure 4.1: An incoming node v_+ attaches to each of the existing nodes v_i for $i \in 1 \dots 4$ with probability p_i. If an edge (v_i, v_p) is formed, its corresponding weight is w_i.

construction approaches in [28, 30]. The starting point is the user-item or rating matrix, which we will refer to as $\mathbf{X} \in \mathbb{R}^{U \times I}$. Following common convention, the missing entries $(u, i) \in \mathbf{X}$ will be set to 0. We select a subset $\mathcal{V}$ of N users which we call the existing users. We compute similarities between these existing users using their ratings vectors. These similarities are then collected in the user similarity matrix $\mathbf{B} \in \mathbb{R}^{U \times U}$ which is a symmetric matrix with 1s on the main diagonal. The matrix $\mathbf{B}$ can be considered the adjacency matrix of a fully connected undirected graph, where $[\mathbf{B}]_{uv}$ denotes the similarity between the user u and v. To build an existing graph with an adjacency matrix $\mathbf{B}^i$ specific for an item i, we remove any edge from a user for which it holds that $[\mathbf{X}]_{ui} = 0$, in other words, we remove edges from users who did not rate the item i. The rationale behind this is that users who did not rate the item should not affect the rating predictions of others. The corresponding graph signal is $\mathbf{x}^i = [[\mathbf{X}]_{1i}, \dots, [\mathbf{X}]_{Ni}]$, which contains the ratings of item i by all existing users. We consider the remaining users as incoming users whose attachment patterns are unknown. We use stochastic attachment to attach them to the existing graph and obtain the stochastically expanded graph $\mathcal{G}^i_+ = (\mathcal{V}_+, \mathcal{E}_+)$ with adjacency matrix $\mathbf{B}^i_+$.

Specifically, we use heuristic stochastic attachment and learned stochastic attachment.

4.3. Graph Signal Interpolation

Graph filters and Graph Convolutional Neural Networks are instruments to process a graph signal while accounting for its coupling with the graph topology [18]. For clarity, we generalize graph processing models that use graph convolutions (such as graph filters and GCNNs) as a graph convolution model, which is a mapping of the form $\Phi(\mathbf{x}; \mathbf{S}; \mathcal{H})$ characterized by a graph signal $\mathbf{x}$, a graph shift operator $\mathbf{S}$, and a set of coefficients $\mathcal{H}$, which we abbreviate to $\Phi(\cdot)$. We can use a graph convolution model to interpolate the graph at the incoming node v_+ for which the graph signal x^i_+ is unknown. To this end, we define the graph signal of the stochastically expanded graph $\mathcal{G}^i_+$ as $\mathbf{x}^i_+ = [\mathbf{x}^{i\top}, 0]^\top$. Next, we apply $\Phi(\cdot)$ to obtain the graph convolution output $\mathbf{y}_+^i$. This operation is formulated as

$$\mathbf{y}_+^i = \Phi(\mathbf{x}_+^i, \mathbf{B}_+^i, \mathscr{H}). \tag{4.4}$$

If we instantiate Φ as a graph filter, we can particularize the graph convolution output in eq. (4.4) as

$$\mathbf{y}_+^i = \mathbf{H}(\mathbf{B}_+^i)\mathbf{x}_+^i = \sum_{k=1}^{K} h_k \left[\mathbf{B}_+^i\right]^k \mathbf{x}_+^i, \tag{4.5}$$

where $\mathbf{H}(\mathbf{B}_+^i)$ is the filtering matrix $\sum_{k=1}^{K} h_k \left[\mathbf{B}_+^i\right]^k$ with $\mathbf{h} = [h_1, \ldots, h_L]^\top$ being the K filter coefficients and K the filter order. Note that the definition of graph filters in eq. (2.13) starts the summation at $k = 0$, to incorporate the unshifted graph signal into the processed output. However, since the graph signal at node v_+ is equal to 0, it does not contribute to the interpolated signal and therefore $k = 0$ is omitted.

If, instead, we instantiate Φ as a convolutional neural network of L layers, the output of each layer is defined as

$$\mathbf{x}_l^f = \sigma\left[\sum_{g=1}^{F_{l-1}} \sum_{k=1}^{K} h_{kl}^{fg} \left[\mathbf{B}_+^i\right]^k \mathbf{x}_{l-1}^g\right] = \left[\sum_{g=1}^{F_{l-1}} \mathbf{h}_l^{fg}(\mathbf{B}_+^i)\mathbf{x}_{l-1}^g\right], \text{ for } f = 1, \ldots, F_L. \tag{4.6}$$

In the final GCNN layer L we end up with F_L features $\mathbf{x}_L^1, \ldots \mathbf{x}_L^{F_L}$. To obtain a scalar output for each node n, we feed the different features $[x_{Ln}^1, \ldots, x_{Ln}^{F_L}]$ into a fully connected layer, which is a matrix of size $F_L \times 1$. The output of this fully connected layer for all nodes combined is the output graph signal $\mathbf{y}_+^i$ of the GCNN. The predicted rating is then $\hat{x}_+^i = [\mathbf{y}_+^i]_{v_+}$.

4.4. TRAINING GRAPH CONVOLUTION MODEL

The performance of a graph convolution model for recommendation for the incoming node depends on the parameters $\mathscr{H}$. In this work, we consider two tasks: rating and ranking. For rating prediction, we require a training set $\mathscr{T}_{tr}^{rate} = \{(v_+, x_+^*)\}$ containing tuples of the id of an incoming node and his corresponding rating. For ranking, we require a training set $\mathscr{T}_{tr}^{rank} = \{(v_+, i, j)\}$ which contains triplets of the id of an incoming user, an item i, and an item j for which it holds that $i >_{v_+} j$, meaning v_+ prefers item i over j. We use empirical risk minimization to solve for a set of optimal parameters which minimize the training loss. To do so, we define a loss function $\mathscr{L}$ that is appropriate for the task so that we can optimize our coefficients accordingly.

4.4.1. LEARNING TO RATE

The task of learning to rate is concerned with predicting unknown ratings. We use the MSE as a loss function to penalize the difference between the actual and predicted ratings. Formally, we define the MSE loss function as

$$\mathscr{L} = \text{MSE} = ([\Phi(\mathbf{x}_+^i, \mathbf{B}_+^i, \mathscr{H})]_{v_+} - x_+^*)^2, \tag{4.7}$$

with x_+^* being the true rating of v_+. Note that $\mathbf{B}_+^i$ is a stochastic adjacency matrix. Therefore, we must consider the expected MSE, which is defined as

$$\mathscr{L} = \mathbb{E}[\text{MSE}] = ([\Phi(\mathbf{x}_+^i, \mathbb{E}[\mathbf{B}_+^i], \mathscr{H})]_{v_+} - x_+^*)^2. \tag{4.8}$$

To learn to predict the unknown ratings of the stochastically attached nodes, we must minimize the MSE with respect to $\mathscr{H}$. Formally, this optimization problem is defined as

$$\underset{\mathscr{H}}{\text{minimize}} \sum_{(v_+, x_+^*) \in \mathscr{T}_{tr}^{rate}} ([\Phi(\mathbf{x}_+^i, \mathbb{E}[\mathbf{B}_+^i], \mathscr{H})]_{v_+} - x_+^*)^2 + \mu_c \|\mathscr{H}\|_2^2, \tag{4.9}$$

where $\mu_c (> 0)$ is a control variable for L_2-norm regularization of the convolution coefficients to prevent overfitting.

4.4.2. LEARNING TO RANK

Instead of learning to predict unknown ratings, the ranking task addresses the problem of predicting preferences between items by a user. While the shapes of inputs and outputs of the graph convolution model are the same, their interpretations differ following from the difference in task. For the ranking task, we use Bayesian Personalized Ranking (BPR) loss as a loss function [52]. BPR loss uses the preference of an item i over an item j by a user j, in short, $i >_u j$. It maximizes the likelihood of the model parameters Θ based on these observed pairwise preferences. Formally, it maximizes

$$\mathbb{P}(\Theta | i >_u j) \propto \mathbb{P}(i >_u j | \Theta) \mathbb{P}(\Theta), \tag{4.10}$$

where $\mathbb{P}(i >_u j|\Theta)$ is the likelihood function and $\mathbb{P}(\Theta)$ the prior. The likelihood function can be rewritten as

$$\begin{aligned} \mathbb{P}(i >_u j | \Theta) &= \sigma(\hat{r}_{ui} - \hat{r}_{uj}) \\ \text{with} \quad \sigma &= \frac{1}{1 + e^{-x}}. \end{aligned} \tag{4.11}$$

We take the natural logarithm of the logistic sigmoid σ to arrive at the final loss function defined as

$$\mathscr{L} = - \sum_{(v_+, i, j) \in \mathscr{T}_{tr}^{rank}} \ln \sigma([\Phi(\mathbf{x}_+^i, \mathbb{E}[\mathbf{B}_+^i], \mathscr{H})]_{v_+} - [\Phi(\mathbf{x}_+^j, \mathbb{E}[\mathbf{B}_+^j], \mathscr{H})]_{v_+}). \tag{4.12}$$

Note that we negate the sum because we need a loss function, that is a function that is low for accurate predictions and vice versa. In the context of ranking, an accurate prediction maximizes the difference in predicted ratings between an item i preferred over an item j. The ranking optimization problem can be formulated accordingly as

$$\underset{\mathscr{H}}{\text{minimize}} \quad - \sum_{(v_+, i, j) \in \mathscr{T}_{tr}^{rank}} \ln \sigma([\Phi(\mathbf{x}_+^i, \mathbb{E}[\mathbf{B}_+^i], \mathscr{H})]_{v_+} - [\Phi(\mathbf{x}_+^j, \mathbb{E}[\mathbf{B}_+^j], \mathscr{H})]_{v_+}) + \mu_c \|\mathscr{H}\|_2^2. \tag{4.13}$$

Note that we use the observed ratings to deduce item preferences, but we do not use the actual values in this optimization problem.

4.5. LEARNING STOCHASTIC ATTACHMENT

Whereas the previously stated optimization problems assume that we have fixed values for **p** and **w**, we can also optimize the stochastic attachment model to minimize the task-specific loss. To this end, we consider **p** and **w** in Equation (4.3) to be vectors containing trainable probabilities and edge weights, respectively. Consequently, we train the graph convolution coefficients and the stochastic model so that the model learns which existing nodes are important for the specific task. Important existing nodes will have a high probability of forming an edge with learned weight to the incoming node.

Instead of minimizing task-specific losses in Equation (4.9) and Equation (4.13) by only solving for the graph convolution coefficients $\mathscr{H}$, we jointly optimize those coefficients along with the parameters in **p** and **w**. For rating, this is formalized as the joint optimization problem

$$\begin{aligned}\underset{\mathscr{H},\mathbf{p},\mathbf{w}}{\text{minimize}} \quad & \sum_{(v_+,x_+^i)\in\mathscr{T}} ([\Phi(\mathbf{x}_+^i,\mathbb{E}[\mathbf{B}_+^i],\mathscr{H})]_{v_+} - x_+^i)^2 + \mu_c\|\mathscr{H}\|_2^2 - \mu_p\|\mathbf{p}\circ\mathbf{w}\|_2^2 \\ \text{subject to} \quad & \mathbf{p}\in[0,1]^N, \mathbf{w}\in[0,1]^N,\end{aligned} \tag{4.14}$$

where μ_p is the control variable for the L_2 norm regularization of the attachment pattern coefficients to reduce its sparsity. Note that **p** and **w** are embedded in $\mathbb{E}[\mathbf{B}_+^i]$. For the ranking task, we state the joint optimization problem analogously as

$$\begin{aligned}\underset{\mathscr{H},\mathbf{p},\mathbf{w}}{\text{minimize}} \quad & -\sum_{(v_+,i,j)\in\mathscr{T}} \ln\sigma([\Phi(\mathbf{x}_+^i,\mathbb{E}[\mathbf{B}_+^i],\mathscr{H})]_{v_+} - [\Phi(\mathbf{x}_+^j,\mathbb{E}[\mathbf{B}_+^j],\mathscr{H})]_{v_+}) \\ & + \mu_c\|\mathscr{H}\|_2^2 - \mu_p\|\mathbf{p}\circ\mathbf{w}\|_2^2 \\ \text{subject to} \quad & \mathbf{p}\in[0,1]^N, \mathbf{w}\in[0,1]^N.\end{aligned} \tag{4.15}$$

We optimize both of these problems too using a gradient descent-based approach.

4.6. BASELINES

In this work, we compare several non-graph and graph-based methods that perform pure cold start recommendation. In this section we present all baselines that we use in this comparison. We will first present our four graph-based methods, and after that the two non-graph baselines.

1. TRUE CONNECTIVITY

Using the true connectivity derived from rating similarity, we attach a node analogously to the construction of the existing graph. Specifically, we calculate similarities between all existing users in $\mathscr{V}$ and the incoming user v_+ using the observed ratings. Edges are formed between the top-k users that are most similar to v_+, with an edge weight equal to the corresponding similarity. Although this baseline does not represent a pure cold start scenario, since it uses rating data from v_+, it is useful to get an idea of how far our method is from this non-cold start scenario.

2. FIXED GRAPH CONVOLUTION MODEL, HEURISTIC-BASED ATTACHMENT

A naive approach to ameliorating the pure cold start problem using graphs is to attach v_+ using some stochastic attachment model based on heuristics. This model defines the probability that an existing node connects to v_+ using heuristics. Then we apply a graph convolution model that is trained on existing graphs on the expanded graph to interpolate the graph signal at v_+. In this work, we use two important random graph models to infer attachment, the Erdős-Rényi(ER) and Barabási-Albert(BA) models [4, 16]. The ER model randomly forms an edge between an existing node and v_+ following a uniform distribution that is the same for all nodes. We set the probability to be k divided by the number of existing nodes N, so that, in expectation, k edges are formed. Using the ER model, the probabilities $p_i \in \mathbf{p}$ are set to

$$p_i = \frac{k}{N}, \tag{4.16}$$

and the weights $w_i \in \mathbf{w}$ to μ_i, which is the mean weight of the outgoing edges for each node.

The BA model randomly forms an edge between an existing node and v_+ following a uniform distribution that differs per node. The probability that a node $v_i \in \mathcal{V}$ connects to v_+ is proportional to the out-degree d_i of v_i, also called preferential attachment. Formally, the probabilities $p_i \in \mathbf{w}$ are set to

$$p_i = \frac{d_i}{\sum_{j=1\ldots V_e} d_j}, \tag{4.17}$$

and the weights $w_i \in \mathbf{w}$ to μ_i.

3. LEARNABLE GRAPH CONVOLUTION MODEL, HEURISTIC-BASED ATTACHMENT

Instead of using a pretrained graph convolution model, we train it on graphs that are stochastically expanded using one of the heuristic models in Section 4.6. When doing so, training and testing situations are analogous, which is expected to result in better performance.

4. LEARNABLE GRAPH CONVOLUTION MODEL, LEARNABLE STOCHASTIC ATTACHMENT

As presented in Section 4.1, we can also use a learnable stochastic attachment model that we optimize jointly with the graph convolution model. By doing so, we try to fit the stochastic model to the recommendation tasks.

5. MEAN RATING

A very basic approach is to use the mean rating of an item as the predicted rating for a pure cold start user. For items with a considerate number of ratings, the mean rating prediction is much more stable than for items with only a few ratings.

6. SVD++

SVD++ is an important algorithm in the world of recommendation systems. It uses a combination of explicit and implicit ratings [37]. Although the name resembles Singular Value Decomposition (SVD), its mechanics are quite different. It predicts an unknown rating of user u to item i using the formula

$$\hat{r}_{ui} = \mu + b_i + b_u + \mathbf{q}_i^\top \left(\mathbf{p}_u + |\mathcal{R}_u|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}_u} \mathbf{y}_j \right), \tag{4.18}$$

where μ is the mean of all observed ratings, b_i and b_u are the means of all observed ratings of i and u, and $\mathbf{q}_i, \mathbf{p}_i, \mathbf{y}_j$ are latent factor vectors of dimension f. $\mathscr{R}_u$ denotes the set items rated by user u. Note the analogy with our graph-based methods, as it uses the representations of neighboring items of i to predict its rating. SVD++ does, however, not support pure cold start user prediction by default. Therefore, we modify the implementation in [29] by selecting the top S users $\mathscr{U}_s$ with the most observed ratings and aggregate them into one surrogate user u_s. Specifically, we aggregate over the selected users to get

$$b_{u_s} = \frac{1}{|\mathscr{U}_s|} \sum_{u \in \mathscr{U}_s} b_u, \tag{4.19}$$

and

$$\mathbf{p}_{u_s} = \frac{1}{|\mathscr{U}_s|} \sum_{u \in \mathscr{U}_s} \mathbf{p}_u. \tag{4.20}$$

Furthermore, we define

$$\mathscr{R}_{u_s} = \bigcap_{u \in \mathscr{U}_s} \mathscr{R}_u, \tag{4.21}$$

meaning that the set of items rated by our surrogate user is the intersection of all sets of items rated by the users in $\mathscr{U}_s$.

4.7. OPTIMIZING WITH GRADIENT DESCENT

To solve the optimization problems in eq. (4.9), eq. (4.13), eq. (4.14) and eq. (4.15), we use a gradient descent-based approach. This approach computes the derivatives with respect to the model's learnable parameters using backpropagation [2]. These parameters are updated proportional to these derivatives. For the SVD++ baseline, these parameters are $\mathbf{b}, \mathbf{u}, \mathbf{Q}, \mathbf{P}, \mathbf{Y}$. For the graph-based approaches, in the cases of a learnable stochastic attachment model, these parameters include $\mathbf{p}$ and $\mathbf{w}$. Otherwise, only the graph convolution model parameters are optimized.

4.8. CONCLUSION

This chapter introduced the method proposed in this thesis. To summarize, we propose to compare several graph-based and non-graph baselines to the proposed stochastic attachment models. We proposed two ways of using stochastic attachment for pure cold start recommendation. The first was to train a graph convolution model on graphs that are expanded using a heuristic attachment model, specifically Barabási-Albert and Erdős-Rényi. The second was to jointly train the graph convolution model and a stochastic attachment model. We identified the rating and ranking tasks to train. For both of these tasks, we will evaluate and compare the results of the methods in Section 4.6, which will provide us the answers to the research questions.

5 Numerical Results

This chapter will describe the experiments and results in detail. We evaluate and compare the results among different attachment methods and baselines for different datasets. In Section 5.1 we describe the datasets used for the experiments. Section 5.2 covers the different types of graph convolution model we use in our experiments. In Section 5.3, we describe the general setup of the experiments. Section 5.4 present the results of the experiments. In Section 5.5 we perform sensitivity analysis. We conclude the chapter in Section 5.6 by discussing the observations.

5.1. DATASETS

We conduct the experiments on three well-known datasets: Jester [19], MovieLens-1M [22], and Yelp[1]. These datasets have different levels of size and sparsity, which will provide us with more insight into the performance of the proposed method. Table 5.1 summarizes the statistics for each dataset.

Dataset	**Jester**	**MovieLens-1M**	**Yelp**
No. Users	59132	6040	1311199
No. Items	140	3706	149609
No. Ratings	1761439	1000209	3429186
Sparsity	80.41%	95.31%	99.99%
Rating scale	[-10, 10]	[1,5]	[1,5]

Table 5.1: Statistics from the Jester, MovieLens-1M, and Yelp datasets. Sparsity is defined as the proportion of unobserved ratings in the dataset. Ratings are integers.

5.2. GRAPH CONVOLUTION MODELS

The proposed graph methods use graph convolution models to diffuse the graph signal over the expanded topology. We already introduced two models: graph filters and GCNNs. To recapitulate, a graph filter linearly combines the K-shifted versions of the graph signal. A GCNN is composed of multiple sequential layers of parallel graph filters, with nonlinear activation functions in between the layers. The most primitive type of graph convolution model is k-Nearest Neighbor (kNN), which performs a weighted sum of the graph signal of the direct neighbors of a target node. We formulate the kNN operator as

$$\mathbf{y} = \mathbf{S}\mathbf{x}, \tag{5.1}$$

with $\mathbf{S}$ being the graph shift operator, and $\mathbf{x}$ and $\mathbf{y}$ the input and output graph signal, respectively. Note that this operator is equivalent to a graph filter of order 1, where h_0 is equal to 0 and h_1 is equal to 1. Unlike graph filters and GCNNs, it has no learnable parameters. So, for the graph-based approaches, we employ the following graph convolution models:

1. Graph Filter

2. Graph Convolutional Neural Network

3. kNN

This will shed light on the effect of propagating the signal through the multi-hop neighborhood and learning the corresponding coefficients.

[1] https://www.yelp.com/dataset

5.3. EXPERIMENTAL SETUP

All experiments follow the graph construction procedure as described in Section 4.2. We use $N = 300$ randomly selected users as existing users. The remaining users are split $80\% - 20\%$ into training and testing users, which behave as pure cold starters. We do 5 different realizations to get a more objective insight. We remove 50% of the items that we used to deduce the true attachment patterns to separate them from the ratings we used for testing. We use Bayesian Optimization to perform hyperparameter optimization for the model parameters [74]. For computational reasons, we only use the MovieLens-1M dataset and used the results for all experiments. For the graph filter, this results in $k = 35$ for the kNN graphs and a filter order of 7. The optimal GCNN architecture is 3 layers, each of 11 features and filter banks of order 4. For SVD++, the optimal number of latent factors is 20. The remaining hyperparameters that are not part of the model are set according to the literature and used for all experiments. These are the learning rate $l = 5e-4$, the number of epochs $n = 20$, and the batch size $b = 8$. We train the parameters using the ADAM optimizer [35]. To evaluate the models trained for rating, we use MAE and RMSE as metrics. For the models trained for ranking, we used NDCG, mAP, and recall.

5.4. RESULTS

In this section, we cover the research questions and present the supporting results to answer them. The research questions are as follows.

RQ.1 How can we alleviate the pure cold start problem in collaborative filtering via graph convolutions by training over stochastically expanded graphs?

RQ.2 Can we jointly learn a stochastic attachment model and graph convolutions instead of relying on a heuristic attachment model?

We divide the items in each data set into three sets according to their number of observed ratings: low ($\mathscr{L}$), medium ($\mathscr{M}$), and high ($\mathscr{H}$). We need two thresholds t_l and t_m to define these sets. Accordingly, we define the item sets as $\mathscr{L} = \{i|R(i) \leq t_l\}$, $\mathscr{M} = \{i|t_l < R(i) \leq t_m\}$, and $\mathscr{H} = \{i|t_m < R(i)\}$, for which $R(i)$ is the number of observed ratings for each item. We determine the values of t_l and t_m using histograms of the number of ratings per item, which can be found in Appendix C. We use $t_l = 60, t_m = 100$ for Jester, $t_l = 20, t_m = 100$ for MovieLens-1M, and $t_l = 10, t_m = 20$ for Yelp. Furthermore, we distinguish between the different types of graph convolution model.

RQ.1

Our first research question comprises two parts. The first is to research whether using stochastic attachment results in better performance compared to non-graph baselines. The second is to investigate whether training over stochastically expanded graphs is beneficial compared to training on existing graphs and using induction to test it on stochastically expanded graphs.

For the first part, we train and test graph convolution models on each type of attachment: Barabási-Albert, Erdős-Rényi, true, and learned attachment. We compare with

		Item category			
Dataset	**Method**	**All (95% CI)**	**Low (95% CI)**	**Medium (95% CI)**	**High (95% CI)**
Jester	True att.	2.7818 [2.7623 - 2.8015]	5.4810 [4.8295 - 6.1232]	4.2002 [3.3648 - 5.0791]	2.7733 [2.7539 - 2.7927]
	Mean	4.0975 [4.0714 - 4.1230]	5.0231 [4.4339 - 5.5994]	**4.1592 [3.3458 - 5.0179]**	4.0957 [4.0699 - 4.1215]
	SVD++	4.2798 [4.2658 - 4.2940]	**4.2373 [3.7294 - 4.7418]**	4.1777 [3.3298 - 5.0718]	4.2799 [4.2659 - 4.2938]
	BA att.	4.3207 [4.2946 - 4.3476]	4.3985 [3.8555 - 4.9419]	4.2762 [3.5106 - 5.0889]	4.2748 [4.2473 - 4.3027]
	ER att.	4.1101 [4.0846 - 4.1361]	4.6523 [4.1331 - 5.1537]	4.2363 [3.4647 - 5.0532]	4.1085 [4.0830 - 4.1341]
	Learned	**4.0903 [4.0644 - 4.1166]**	4.3056 [3.8360 - 4.7679]	4.2224 [3.4273 - 5.0449]	**4.0897 [4.0639 - 4.1151]**
ML-1M	True att.	0.7033 [0.7000 - 0.7066]	1.0698 [0.8516 - 1.3013]	0.7824 [0.7761 - 0.7886]	0.6605 [0.6567 - 0.6642]
	Mean	0.9547 [0.9503 - 0.9592]	1.1456 [0.9402 - 1.3563]	0.9705 [0.9633 - 0.9776]	0.9467 [0.9411 - 0.9522]
	SVD++	0.9458 [0.9418 - 0.9497]	1.1342 [0.9255 - 1.3429]	0.9631 [0.9562 - 0.9699]	0.9369 [0.9323 - 0.9416]
	BA att.	0.9408 [0.9366 - 0.9450]	**1.0645 [0.8888 - 1.2380]**	0.9652 [0.9583 - 0.9719]	0.9284 [0.9232 - 0.9335]
	ER att.	**0.9402 [0.9362 - 0.9444]**	1.0972 [0.8981 - 1.3003]	**0.9649 [0.9581 - 0.9717]**	**0.9277 [0.9225 - 0.9330]**
	Learned	0.9474 [0.9434 - 0.9514]	1.1238 [0.9015 - 1.3510]	0.9750 [0.9681 - 0.9820]	0.9334 [0.9285 - 0.9383]
Yelp	True att.	0.7426 [0.7366 - 0.7485]	0.8097 [0.8002 - 0.8195]	0.7331 [0.7225 - 0.7438]	0.6416 [0.6316 - 0.6514]
	Mean	0.9525 [0.9456 - 0.9593]	0.9885 [0.9776 - 0.9991]	0.9383 [0.9259 - 0.9508]	0.9126 [0.8997 - 0.9254]
	SVD++	0.9327 [0.9261 - 0.9392]	0.9706 [0.9601 - 0.9811]	0.9157 [0.9038 - 0.9273]	0.8932 [0.8810 - 0.9054]
	BA att.	**0.9031 [0.8938 - 0.9125]**	**0.9472 [0.9244 - 0.9703]**	**0.9002 [0.8844 - 0.9160]**	0.8884 [0.8753 - 0.9016]
	ER att.	0.9034 [0.8941 - 0.9127]	0.9498 [0.9263 - 0.9736]	0.9011 [0.8853 - 0.9168]	**0.8876 [0.8746 - 0.9007]**
	Learned	0.9226 [0.9157 - 0.9292]	0.9524 [0.9420 - 0.9629]	0.9011 [0.8853 - 0.9168]	0.8896 [0.8775 - 0.9016]

Table 5.2: RMSE of all items in all datasets. Bold font indicates the best result per item category for each dataset. Lower is better. The graph-based approaches all used a graph filter ($k = 7$). The attachments in the Method column denote the type of attachment used for both training and testing. All values can be found in Appendix A.

the non-graph baselines: mean rating and SVD++. Table 5.2 shows the RMSE rating test scores of these methods on all datasets. We computed confidence intervals using bootstrapping. We only included the results obtained by the graph filter for the sake of clarity. Complete rating results can be found in Appendix A and ranking results in Appendix B. In general, we see that the graph filter and GCNN outperform the kNN model, which corroborates our hypothesis that using the multi-hop neighborhood improves prediction accuracy. For MovieLens-1M and Yelp, the true attachment baseline outperforms the other baselines by a large margin (17.8% – 32%). Note, however, that this uses rating data on the incoming user, which is not available in a pure cold start setting. The graph-based approaches outperform the non-graph baselines by a small margin. Specifically, for the MovieLens-1M and Yelp datasets, the BA method significantly outperforms the SVD++ baseline ($p < 4e-9$ and $p = 0.0001$, respectively). In particular, in the low-item category, the Barabási-Albert model outperforms the others. This can be explained by the construction of existing graphs in combination with the essence of preferential attachment used by the Barabási-Albert model. We will recapitulate this procedure. Each node u in the existing graph is connected to an existing node v if $u \in \mathcal{K}_v$, where $\mathcal{K}_v$ is the set of the top-k existing nodes most similar to u. This implies that an existing node with either a lot of ratings or a "general" rating behavior will have more outgoing edges, in other words, a higher out-degree. Since the Barabási-Albert attachment probabilities are proportional to a node's out-degree, the probability of an incoming node connecting to an existing node with many ratings is larger. This can be beneficial for items with only a few ratings, as in general users with a lot of ratings better predict unknown ratings than users with fewer ratings.

The ranking results show small differences between all methods, as shown in Ap-

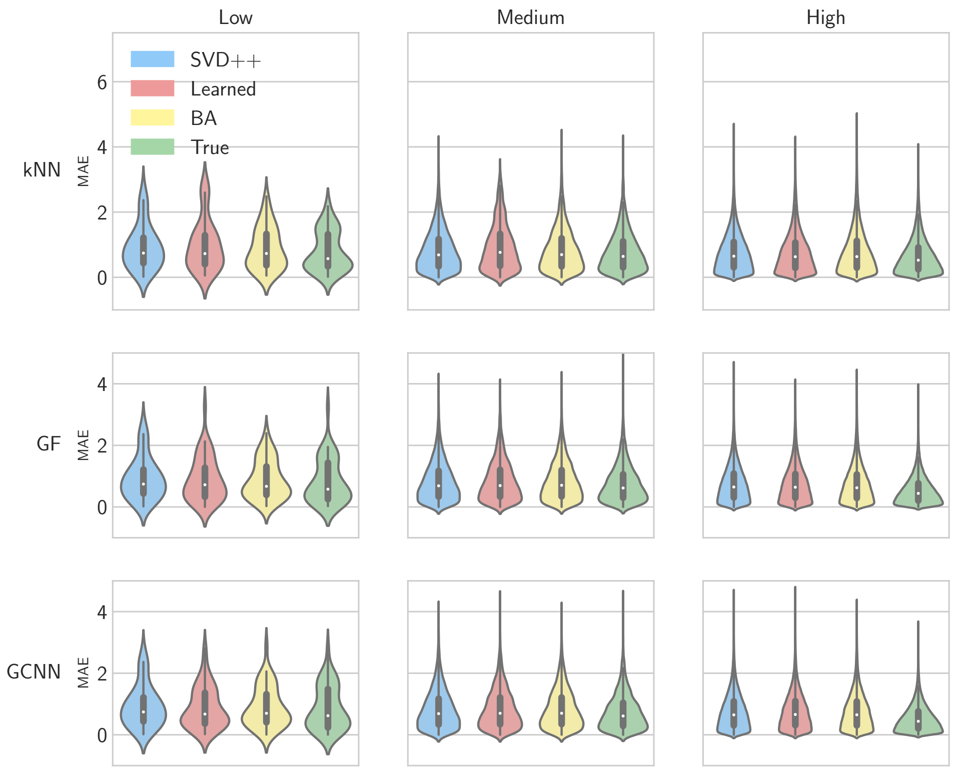
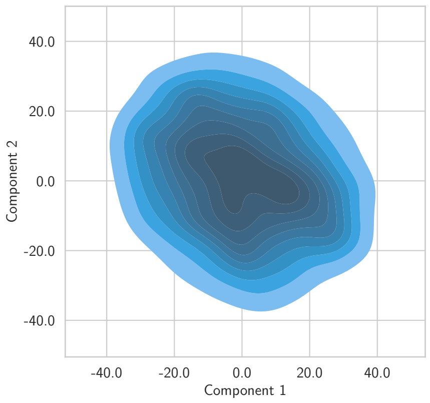


Figure 5.1: MAE violin plots of four models trained and tested on MovieLens-1M: Mean rating (blue), SVD++ (red), learned stochastic attachment (yellow) and true attachment (green). Each row is for a different type of graph convolution which is denoted on the left. Each column denotes the item category, which is denoted at the top. The raw evaluation metrics are in Appendix A.

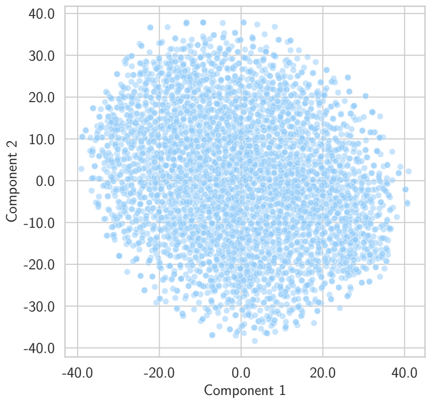
pendix B. This is believed to be caused by the nature of ranking prediction. The difference in ranking 'score' must be substantial to cause a shift in the ranking. For example, say that a ranking model A predicts 1.0 and 1.5 for items i and j respectively. Then assume that another model B predicts 1.45 and 1.5 for items i and j. Although the results for item i differ significantly (+45%), the ranking is equal ($B > A$) and consequently the ranking evaluation metrics will also produce the same results.

Figure 5.1 shows the MAE violin plots of these models trained and tested on the MovieLens-1M dataset. We only show the best heuristic model, which was the Barabási-Albert model. We see that for the low category, the Barabási-Albert trained graph filter performs the best and that its error distribution is also more dense at lower errors. This indicates that this method is more robust when only a few ratings are available than the non-graph baselines. For the medium and high categories, the differences are smaller. This is remarkable, as SVD++ predicts ratings deterministically by creating a surrogate user of 10 randomly selected existing users.

Figure 5.2 shows a two-dimensional embedding of the latent factors of all MovieLens-



(a) KDE plot of t-SNE embedding



(b) Scatter plot of t-SNE embedding

Figure 5.2: Visualization of the two-dimensional embedding of the latent factors of MovieLens-1M users learned by the SVD++ algorithm. The embedding is calculated using t-SNE with a perplexity of 40.

1M users, as learned by SVD++. For Jester and Yelp, these plots can be found in Appendix C. These embeddings appear to be jointly normally distributed around $(0,0)$ without many outliers. Therefore, taking the mean of multiple randomly selected vectors of latent factors will always result in approximately the same aggregated vector. Due to the fact that these vectors are normally distributed, the true latent factor vector of a pure cold start user will on average be close to this mean vector, which explains the relatively high accuracy of this baseline. Another explanation for the relatively high accuracy of this baseline is that graph-based methods train the graph convolution model to propagate the graph signal so that it can predict the unknown rating of the incoming user. To do so, it needs the graph signal of the existing graph. This signal comprises at most 300 data points, since this is the size of the existing graph. SVD++, on the other hand, uses all training data to learn embeddings of users and items, which is a lot more. Therefore, it has an advantage over the graph methods. To further investigate the effect of the size of the existing graph on performance, we built the existing graphs of 600 users for MovieLens-1M and the trained and tested a graph filter using the true, Barabási-Albert, Erdős-Rényi and learned attachments. The results of these experiments are shown in Table 5.3. Whereas the learned attachment performs the worst of all graph methods on the smaller graphs, it performs the best of all graph methods on the larger graphs. It seems that this method needs more data than it gets from the existing graph of 300 users to achieve satisfactory accuracy.

Another remarkable result shown in Table 5.2 is that for the Jester dataset, the learned stochastic attachment model shows good performance compared to the other datasets, and even compared to true attachment. For low-category items, it even outperforms the true attachment baseline. Note that using a GCNN and learned stochastic attachment, the performance on low items was slightly better than SVD++ as shown in Appendix A. The Jester dataset is quite different from the other datasets, as it only has a few items and is relatively dense. Therefore, the learned stochastic model has to generalize over

	Item category			
Attachment	**All (95% CI)**	**Low (95% CI)**	**Medium (95% CI)**	**High (95% CI)**
True	0.6528 [0.6494 - 0.6562]	1.1275 [0.8975 - 1.3731]	0.7165 [0.7104 - 0.7227]	0.6127 [0.6090 - 0.6166]
BA	0.9431 [0.9389 - 0.9474]	1.1022 [0.9163 - 1.3107]	0.9622 [0.9553 - 0.9690]	0.9319 [0.9266 - 0.9372]
ER	0.9397 [0.9352 - 0.9441]	1.0930 [0.9066 - 1.3004]	0.9560 [0.9490 - 0.9631]	0.9300 [0.9244 - 0.9356]
Learned	**0.9341 [0.9298 - 0.9384]**	**1.0776 [0.8957 - 1.2754]**	**0.9524 [0.9455 - 0.9594]**	**0.9234 [0.9181 - 0.9289]**

Table 5.3: RMSEs of a graph filter ($k = 7$) trained and tested on all items in the MovieLens-1M dataset using an existing graph of 600 users. The attachments in the method column denote the type of attachment used for both training and testing. Lower is better for all metrics.

a smaller amount of different graph structures, causing it to fit better to the data. This explains its remarkable performance. Due to its relative density, SVD++ and mean rating also perform well. Each item has many ratings which is believed to be beneficial for the performance. The reason why the mean rating outperforms SVD++ in the high category may be that the latter uses items rated by the target user. Since there are only a few items in this dataset, this procedure is less effective since only a few items can be considered when making a prediction.

For the second part, we must find out whether training and testing on stochastically expanded graphs is better in terms of accuracy than training on existing graphs and testing on stochastically expanded graphs. To this end, we train a graph filter on existing graphs on each dataset: one on graphs obeying Barabási-Albert expansion, and one on graphs obeying Erdős-Rényi. We then test four cases: (i) train on ER, test on ER (ii) train on existing, test on ER (iii) train on BA, test on BA (iv) train on existing, test on BA. The models trained on existing graphs use induction to make predictions on the expanded graphs. The MAE distributions for Jester are shown in Figure 5.3, and the corresponding RMSE scores are found in Table 5.4. For all items, the trained ER and BA models outperform the inductive models ($p = 3e-6$, $p = 0.04$, respectively). For items in the low category, the Barabási-Albert and Erdős-Rényi trained and tested graph filters are more robust because the error density is higher at low values. For the other categories, the differences are smaller, but still apparent. We observed the same pattern for the other datasets. This confirms our hypothesis that training on stochastically attached nodes improves the prediction performance. By doing so, the model adapts to this type of attachment and is better able to predict ratings using such a graph compared to the use of a pretrained model on a stochastically expanded graph.

	Item category			
Attachment	**All**	**Low**	**Medium**	**High**
ER trained	4.1101 [4.0843 - 4.1357]	4.6494 [4.1219 - 5.1568]	4.2384 [3.4594 - 5.0521]	4.1088 [4.0836 - 4.1350]
ER tested	4.2083 [4.1807 - 4.2354]	5.1244 [4.5306 - 5.6978]	4.1957 [3.3152 - 5.1252]	4.2069 [4.1802 - 4.2342]
BA trained	4.3209 [4.2951 - 4.3471]	4.4010 [3.8526 - 4.9436]	4.2811 [3.5310 - 5.0736]	4.3206 [4.2943 - 4.3471]
BA tested	4.3698 [4.3417 - 4.3981]	5.1534 [4.5735 - 5.7338]	4.1751 [3.3157 - 5.0974]	4.3687 [4.3406 - 4.3962]

Table 5.4: RMSE graph filters trained and/or tested on Erdős-Rényi and Barabási-Albert expanded graphs of the Jester dataset.

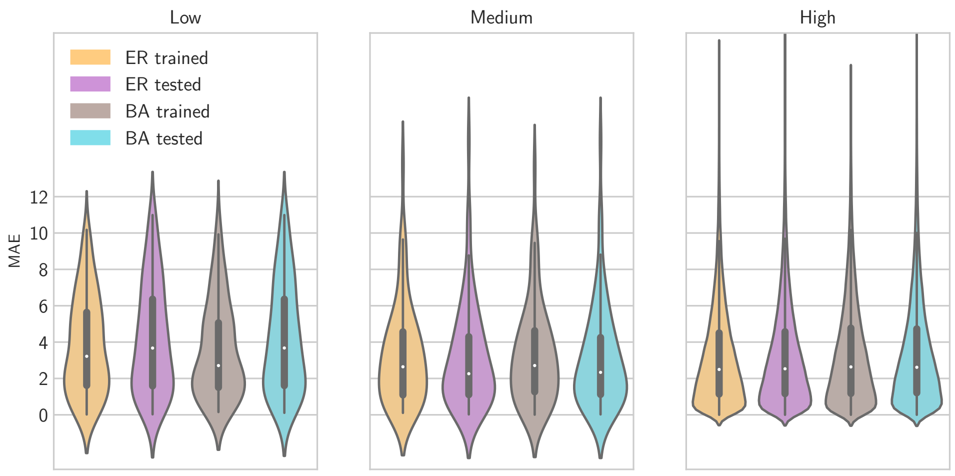


Figure 5.3: MAE violin plots of four graph filters trained and tested on Jester. In the legend. BA or ER trained means trained and tested on expanded graphs using the respective heuristics. BA or ER tested means trained on existing graphs and tested on expanded graphs using the respective heuristics. Each column denotes the category of items, which is denoted at the top.

RQ.2

For the second question, we will provide results that show the differences in performance between learning a stochastic attachment model instead of using a Barabási-Albert or Erdős-Rényi model. Accuracy scores are in Table 5.2. We see that the learned model performs better than the heuristic ones only for the Jester dataset ($p = 0.0001$). For the MovieLens-1M and Yelp datasets, the learned stochastic attachment model becomes relatively better for less sparse items. A reason behind this can be that there are more dense items and that dense items consequently have more incoming users to train on compared to sparse items. This can cause the model to overfit to dense items. If we look at the results from the Jester dataset, we see that the learned attachment model performs the best overall and specifically for the high category. Since this dataset has few items and each item has many ratings, the model has sufficient data to effectively learn the stochastic model. On top of that, due to the fact that the dataset comprises only a few items, the model can generalize better over all items compared to the other datasets that comprise many items. So for dense datasets with a few items, the learned stochastic model is effective, otherwise the heuristic-based models are prefered. Again, for ranking, there were no clear differences.

5.5. SENSITIVITY ANALYSIS

To get an idea of how each model hyperparameter influences the results, we performed a sensitivity analysis on our proposed learnable stochastic model. For the graph filter, we consider the most important parameter, which is the filter order K. For the GCNN, we consider the number of layers and the type of nonlinearity. For the loss function, we consider the scalars μ_c and μ_p. We will consider the same three categories of items as before. For computational reasons, we will only use the Yelp dataset and assume that the outcomes to correspond to the other datasets.

The first hyperparameter we will cover is the term μ_p that controls the $L2$-regularization of the $\mathbf{p} \circ \mathbf{w}$ norm. We trained and tested a graph filter of $K = 7$ for all reported values. The results are found in Table 5.5. We see that the accuracy does not change much when we try different values. The best value appears to be $\mu_p = 1\text{e}-3$.

	$\mu_\mathbf{p}$					
Item cat.	**1e-1**	**1e-2**	**1e-3**	**1e-4**	**1e-5**	**1e-6**
All	0.9231	0.9223	0.9217	0.9230	0.9231	0.9227
Low	0.9521	0.9517	0.9506	0.9518	0.9528	0.9515
Medium	0.9112	0.9102	0.9094	0.9110	0.9108	0.9109
High	0.8913	0.8902	0.8904	0.8918	0.8906	0.8910

Table 5.5: RMSE for different values of μ_p trained and tested on Yelp.

Next, we will try different values for the other $L2$-regularization term μ_c of the graph convolution parameters $\mathscr{H}$. We trained and tested a graph filter of $K = 7$ on learned stochastically expanded graphs for all reported values. The results are found in Table 5.6. Also, for this parameter the outcomes do not change much. The best value is somewhere between $1\text{e}-4$ and $1\text{e}-5$.

	$\mu_\mathbf{c}$					
Item cat.	**1e-1**	**1e-2**	**1e-3**	**1e-4**	**1e-5**	**1e-6**
All	0.9257	0.9238	0.9220	0.9220	0.9219	0.9223
Low	0.9568	0.9534	0.9508	0.9508	0.9508	0.9520
Medium	0.9135	0.9120	0.9098	0.9101	0.9099	0.9101
High	0.8911	0.8908	0.8910	0.8904	0.8905	0.8899

Table 5.6: RMSE for different values of μ_c trained and tested on Yelp.

The most important hyperparameter of a graph filter is the filter order K. We trained and tested graph filters on Barabási-Albert expanded graphs with different values of K. The results are found in Table 5.7. It is clear that using the graph signal from the multi-hop neighborhood improves the accuracy, since the RMSEs for $K = 1$ are significantly higher than the others, particularly for low-category items. This observation corresponds to the literature and our preceding results of the kNN experiments as shown in Table 5.2.

	K					
Item cat.	**1**	**3**	**5**	**7**	**9**	**11**
All	0.9675	0.9208	0.9212	0.9030	0.9216	0.9211
Low	1.0139	0.9486	0.9498	0.9470	0.9493	0.9492
Medium	0.9473	0.9104	0.9106	0.9002	0.9113	0.9104
High	0.9165	0.8892	0.8887	0.8885	0.8902	0.8894

Table 5.7: RMSE for different values of K trained and tested on Yelp.

Now we will move on to the GCNN hyperparameters. We trained and tested a GCNN using different types of nonlinearities on learned stochastically expanded graphs. The results are found in Table 5.8. It appears that the Tanh nonlinear function results in the best accuracy, especially for low-category items.

	Nonlinearity			
Item cat.	**Sigmoid**	**Tanh**	**LeakyReLU**	**ReLU**
All	0.9263	0.9219	0.9235	0.9235
Low	0.9557	0.9495	0.9529	0.9530
Medium	0.9151	0.9113	0.9118	0.9120
High	0.8932	0.8911	0.8910	0.8920

Table 5.8: RMSE for different types of nonlinearities for GCNNs trained and tested on Yelp.

Another hyperparameter specific to GCNNs is the number of layers. Therefore, we trained and tested a GCNN using different number of layers on learned stochastically expanded graphs. The results are found in Table 5.9. It appears that the 7 layers yield the lowest RMSE.

	Number of layers				
Item cat.	**3**	**5**	**7**	**9**	**11**
All	0.9235	0.9253	0.9243	0.9270	0.9288
Low	0.9523	0.9515	0.9510	0.9539	0.9609
Medium	0.9118	0.9167	0.9133	0.9175	0.9160
High	0.8920	0.8946	0.8953	0.8962	0.8932

Table 5.9: RMSE for different numbers of layers for GCNNs trained and tested on Yelp.

5.6. DISCUSSION

In this chapter, we presented the results of the various experiments we carried out for this thesis and covered the main observations.

In Section 5.4 we covered our research questions and the corresponding results. We compared our proposed stochastic methods with two heuristic baselines. For two of the three datasets, the proposed methods outperform those baselines. The performance gap is the largest when an item only has a few ratings, which also reflects a cold start setting accurately, since it is unlikely that a 'cold' recommender system has items with a lot of ratings. Our findings show that the heuristic-based methods outperform the learned stochastic attachment model when there are many items. The reason for that can be that the model is unable to successfully generalize over the various graph structures. The results on the Jester dataset support this hypothesis, as this dataset only comrpises a few items, each with many ratings, and we observed our learned stochastic attachment model to outperform the others. In the general case where we consider all items, we could not significantly determine which of the two heuristical methods is the best. However, for the low category, the Barabási-Albert model was significantly better. Another finding we did not expect is the relatively good performance of our constructed SVD++ baseline. The t-SNE embeddings of the users' latent factors led us to the discovery that taking the mean over a set of users results in a surrogate user that is actually a good approximation of many users in the system, which causes it to perform reasonably well. In summary, our findings suggest that, particularly for sparse items, the use of stochastically expanded graphs improves the accuracy of pure cold start recommendation compared to using deterministic baselines. However, for items with more available ratings, the improvements were smaller. We argue that such cases are rare in a real cold start scenario.

For the second research question, we compared our approach of training on stochastically expanded graphs to only test on them. We observed that this training method resulted in better performance, particularly in the sparse rating setting. Therefore, our findings corroborate the hypothesis that the training approach is indeed advantageous over only testing on stochastically expanded graphs by some margin.

In Section 5.5 we experimented with different settings of the hyperparameters of our models. We saw that using the multi-hop neighborhood indeed improved the accuracy, which was what we expected. We also saw slight differences for different values of the regularization terms.

6 CONCLUSION

In this chapter, we conclude this thesis. In Section 6.1 we answer the research questions posed. We conclude this work in Section 6.2 where we suggest possible directions for future work.

6.1. ANSWERS TO TE RESEARCH QUESTIONS

With our research, we tried to find an answer to the following research questions.

RQ.1 *How can we alleviate the pure cold start problem in collaborative filtering via graph convolutions by training over stochastically expanded graphs?*

RQ.2 *Can we jointly learn a stochastic attachment model and graph convolutions instead of relying on a heuristic attachment model?*

To find answers to these questions, we came up with several baselines in Chapter 4. For the first question, we focused on training on incoming nodes that are attached using heuristic stochastic models Barabási-Albert and Erdős-Rényi. During training, we attached nodes using either models and then trained a graph filter and a GCNN on the stochastically expanded graphs. During testing, we attach an incoming node following the same procedure and use the trained graph convolution model to predict the unknown rating. We showed in Chapter 5 that in some cases we indeed achieve significant improvements compared to the non-graph baselines. To emphasize the training part in the first research question, we conducted an additional experiment that trains a graph filter on stochastically expanded graphs and one on existing graphs. Both were tested on stochastically expanded graphs, which resulted in a relatively large performance gap in favor of the first model. Altogether we have shown that we can indeed alleviate the pure cold start problem with our proposed method, which outperforms baselines by a small margin.

For the second question, we implemented a model that jointly optimizes a graph convolution model and a stochastic attachment model, instead of using a heuristic model. Our experiments showed that only in some cases the learned model outperformed the others. Specifically, the learned model showed to work well on the Jester dataset, which is relatively dense. This led us to the hypothesis that the learned model and the graph models in general suffer from the sparsity in the graph signal, which comprises at most 300 data points. To better explain this, we trained it with the other graph baselines again on a larger existing graph. This showed improvements for all graph baselines. Although the heuristic baselines first outperformed the learned model, on the larger graph, the learned model was better. This confirmed our hypothesis that the graph methods suffer from the graph signal sparsity.

Overall, our findings confirm our hypothesis that we can do pure cold start recommendation by training over stochastically expanded graphs. However, the improvements of the proposed methods, especially the learned stochastic model, were minor in many cases compared to the baselines. We have already given some explanations for this. However, we also want to emphasize the immanent complexity of the problem that we address in this work. We are trying to do pure cold start recommendation without using external information. Specifically, we try to predict ratings of users of which we have no data at all, whereas many approaches in the literature try to circumvent the problem by turning to external data. Therefore, we believe that any improvement to this problem must be considered a relevant contribution.

6.2. FUTURE WORK

We conclude this thesis by providing future work directions that follow from our research. There are three possible branches that we identified: (1) use groups of users as nodes (2) use heterogeneous graphs (3) develop hybrid system with side information.

COMBINING USERS

Our first suggestion is to group users into a single node. We showed that the prediction performance is proportional to the size of the existing graph because more data points are available when testing. However, the train and test times also increase with the size of the existing graph. Therefore, we group similar users into one node. For example, we could select 500 users and combine each group of 5 users most similar to a single node, resulting in a graph of 100 nodes. These nodes are then connected analogously as in this work. Consequently, more data points are available in the existing graph, since we use data from more users. However, the size of the existing graph remains relatively small, preserving the train and test speed.

HETEROGENEOUS GRAPHS

The next possible future direction is to use heterogeneous graphs instead of homogeneous graphs. We can encode the entire rating matrix as a heterogeneous graph and stochastically attach incoming users or items to that graph. Note that users can only stochastically attach to items and vice versa. Consequently, we have more data points in the graph and we support both pure cold start users and items. We can then apply the same graph convolution models on the heterogeneous graph, and we believe this will further improve the recommendations.

HYBRID SYSTEM WITH SIDE INFORMATION

The final possible direction for future research is to develop a hybrid system that combines our proposed method and a side information system. By doing so, the hybrid system can use the available side information and combine it with stochastic attachment for pure cold start users. Consequently, the system supports users with and without side information associated with them. We believe that side information can be beneficial for pure cold start recommendation. Therefore, the more side information available, the more the hybrid system uses it. If there is only little or no side information available, the hybrid system will turn more to the stochastic attachment solution.

Bibliography

[1] Fatemeh Alyari and Jafari Navimipour Nima. Recommender systems: A systematic review of the state of the art literature and suggestions for future research. *Kybernetes*, 47(5):985–1017, January 2018.

[2] Shun-Ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4):185–196, 1993.

[3] Naveen Farag Awad and M S Krishnan. The personalization privacy paradox: An empirical evaluation of information transparency and the willingness to be profiled online for personalization. *Miss. Q.*, 30(1):13–28, 2006.

[4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[5] R Berg, T N Kipf, and M Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[6] Daniel Billsus and Michael J Pazzani. Learning collaborative information filters. https://www.aaai.org/Papers/Workshops/1998/WS-98-08/WS98-08-005.pdf. Accessed: 2022-4-5.

[7] J Bobadilla, F Ortega, A Hernando, and J Bernal. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-Based Systems*, 26:225–238, 2012.

[8] J Bobadilla, F Ortega, A Hernando, and A Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, July 2013.

[9] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Model. User-adapt Interact.*, 12(4):331–370, November 2002.

[10] Ramnath K Chellappa and Raymond G Sin. Personalization versus privacy: An empirical examination of the online consumer's dilemma. *Information Technology and Management*, 6(2):181–202, April 2005.

[11] Lei Chen, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. *Proc. Conf. AAAI Artif. Intell.*, 34(01):27–34, April 2020.

[12] Bishwadeep Das and Elvin Isufi. Learning expanding graphs for signal interpolation. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5917–5921, May 2022.

[13] Delft High Performance Computing Centre (dhpc). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[14] Domo. Data never sleeps 8.0. https://www.domo.com/learn/infographic/data-never-sleeps-8. Accessed: 2022-10-3.

[15] Magnus Ekman. *Learning deep learning: Theory and practice of neural networks, computer vision, natural language processing, and transformers using TensorFlow*. Addison Wesley, Boston, MA, October 2021.

[16] Erdős and Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 1960.

[17] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, WWW '19, pages 417–426, New York, NY, USA, May 2019. Association for Computing Machinery.

[18] Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro. Graphs, convolutions, and neural networks: From graph filters to graph neural networks. *IEEE Signal Process. Mag.*, 37(6):128–138, 2020.

[19] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr. Boston.*, 4(2):133–151, July 2001.

[20] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4):1–19, December 2016.

[21] Bowen Hao, Jing Zhang, Hongzhi Yin, Cuiping Li, and Hong Chen. Pre-Training graph neural networks for Cold-Start users and items representation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 265–273. Association for Computing Machinery, New York, NY, USA, March 2021.

[22] F Maxwell Harper and Joseph A Konstan. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):1–19, December 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. December 2015.

[24] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. LightGCN: Simplifying and powering graph convolution network for recommendation. February 2020.

[25] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining — a general survey and comparison. *SIGKDD Explor. Newsl.*, 2 (1):58–64, June 2000.

[26] Liang Hu, Songlei Jian, Longbing Cao, Zhiping Gu, Qingkui Chen, and Artak Amirbekyan. HERS: Modeling influential contexts with heterogeneous relations for sparse and Cold-Start recommendation. *AAAI*, 33(01):3830–3837, July 2019.

[27] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 263–272. unknown, December 2008.

[28] Weiyu Huang, Antonio G Marques, and Alejandro R Ribeiro. Rating prediction via graph signal processing. *IEEE Trans. Signal Process.*, 66(19):5066–5081, October 2018.

[29] Nicolas Hug. Surprise: A python library for recommender systems. *J. Open Source Softw.*, 5(52):2174, August 2020.

[30] Elvin Isufi, Matteo Pocchiari, and Alan Hanjalic. Accuracy-diversity trade-off in recommender systems via graph convolutions. *Inf. Process. Manag.*, 58(2):102459, 2021.

[31] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A survey on contrastive Self-Supervised learning. *Technologies*, 9(1), 2021.

[32] Ling Jian, Jundong Li, and Huan Liu. Toward online node classification on streaming networks. *Data Min. Knowl. Discov.*, 32(1):231–257, January 2018.

[33] Bowen Jin, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. Multi-behavior recommendation with graph convolutional networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 659–668. Association for Computing Machinery, New York, NY, USA, July 2020.

[34] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, December 2020.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[36] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019.

[37] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, New York, NY, USA, August 2008. Association for Computing Machinery.

[38] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender Systems Handbook*, pages 77–118. Springer US, Boston, MA, 2015.

[39] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Syst. Appl.*, 41(4, Part 2):2065–2073, March 2014.

[40] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A W M van der Laak, Bram van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Med. Image Anal.*, 42:60–88, December 2017.

[41] Siwei Liu, Iadh Ounis, Craig Macdonald, and Zaiqiao Meng. A heterogeneous graph neural model for cold-start recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2029–2032. Association for Computing Machinery, New York, NY, USA, July 2020.

[42] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *IEEE Trans. Knowl. Data Eng.*, pages 1–1, 2021.

[43] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer US, Boston, MA, 2011.

[44] Yuanfu Lu, Yuan Fang, and Chuan Shi. Meta-learning on heterogeneous information networks for cold-start recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 1563–1573, New York, NY, USA, August 2020. Association for Computing Machinery.

[45] Ian MacKenzie, Chris Meyer, and Steve Noble. How retailers can keep up with consumers. https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers, October 2013. Accessed: 2022-10-5.

[46] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. April 2017.

[47] Federico Monti, Michael M Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent Multi-Graph neural networks. April 2017.

[48] Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M Bronstein. Dual-Primal graph convolutional networks. June 2018.

[49] Apurva Narayan and Peter H O'N Roe. Learning graph dynamics using deep neural networks. *IFAC-PapersOnLine*, 51(2):433–438, January 2018.

[50] Tieyun Qian, Yile Liang, Qing Li, and Hui Xiong. Attribute graph neural networks for strict cold start recommendation. *IEEE Trans. Knowl. Data Eng.*, pages 1–1, 2020.

[51] Yanru Qu, Ting Bai, Weinan Zhang, Jianyun Nie, and Jian Tang. An end-to-end neighborhood-based interaction model for knowledge-enhanced recommendation. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, number Article 8 in DLP-KDD '19, pages 1–9, New York, NY, USA, August 2019. Association for Computing Machinery.

[52] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback, 2012.

[53] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B Kantor, editors. *Recommender Systems Handbook*. Springer, Boston, MA, 2011.

[54] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. June 2020.

[55] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. December 2016.

[56] Xiao Sha, Zhu Sun, and Jie Zhang. Hierarchical attentive knowledge graph embedding for personalized recommendation. *Electron. Commer. Res. Appl.*, 48:101071, 2021.

[57] Yue Shi, Martha Larson, and Alan Hanjalic. Collaborative filtering beyond the User-Item matrix: A survey of the state of the art and future challenges. *ACM Comput. Surv.*, 47(1):1–45, May 2014.

[58] Nícollas Silva, Diego Carvalho, Adriano C M Pereira, Fernando Mourão, and Leonardo Rocha. The pure Cold-Start problem: A deep study about how to conquer first-time users in recommendations domains. *Inf. Syst.*, 80:1–12, February 2019.

[59] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009, October 2009.

[60] Jiliang Tang, Suhang Wang, Xia Hu, Dawei Yin, Yingzhou Bi, Yi Chang, and Huan Liu. Recommendation with social dimensions. In *Thirtieth AAAI Conference on Artificial Intelligence*, February 2016.

[61] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. October 2017.

[62] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. DropoutNet: Addressing cold start in recommender systems. *Adv. Neural Inf. Process. Syst.*, 30, 2017.

[63] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Exploring High-Order user preference on the knowledge graph for recommender systems. *ACM Trans. Inf. Syst. Secur.*, 37(3):1–26, March 2019.

[64] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. May 2019.

[65] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *The World Wide Web Conference*, WWW '19, pages 3307–3313, New York, NY, USA, May 2019. Association for Computing Machinery.

[66] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. KGAT: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 950–958, New York, NY, USA, July 2019. Association for Computing Machinery.

[67] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, pages 165–174, New York, NY, USA, July 2019. Association for Computing Machinery.

[68] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. Disentangled graph collaborative filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1001–1010. Association for Computing Machinery, New York, NY, USA, July 2020.

[69] Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. Learning intents behind interactions with knowledge graph for recommendation. In *Proceedings of the Web Conference 2021*, WWW '21, pages 878–887, New York, NY, USA, April 2021. Association for Computing Machinery.

[70] Xiao Wang, Ruijia Wang, Chuan Shi, Guojie Song, and Qingyong Li. Multi-component graph convolutional collaborative filtering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6267–6274, 2020.

[71] Yifan Wang, Suyao Tang, Yuntong Lei, Weiping Song, Sheng Wang, and Ming Zhang. DisenHAN: Disentangled heterogeneous graph attention network for recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, pages 1605–1614, New York, NY, USA, October 2020. Association for Computing Machinery.

[72] Ze Wang, Guangyan Lin, Huobin Tan, Qinghong Chen, and Xiyang Liu. CKAN: Collaborative knowledge-aware attentive network for recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 219–228. Association for Computing Machinery, New York, NY, USA, July 2020.

[73] Tianxin Wei, Ziwei Wu, Ruirui Li, Ziniu Hu, Fuli Feng, Xiangnan He, Yizhou Sun, and Wei Wang. Fast adaptation for Cold-Start collaborative filtering with Meta-Learning. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 661–670, November 2020.

[74] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Dianzi Keji Daxue Xuebao*, 17(1):26–40, 2019.

[75] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. Self-supervised graph learning for recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 726–735. Association for Computing Machinery, New York, NY, USA, July 2021.

[76] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. A neural influence diffusion model for social recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, pages 235–244, New York, NY, USA, July 2019. Association for Computing Machinery.

[77] Le Wu, Junwei Li, Peijie Sun, Richang Hong, Yong Ge, and Meng Wang. DiffNet++: A neural influence and interest diffusion network for social recommendation. *IEEE Trans. Knowl. Data Eng.*, pages 1–1, 2020.

[78] Shiwen Wu, Fei Sun, Wentao Zhang, and Bin Cui. Graph neural networks in recommender systems: A survey. *arXiv:2011.02260 [cs]*, April 2021.

[79] Yuankai Wu, Dingyi Zhuang, Aurelie Labbe, and Lijun Sun. Inductive graph neural networks for spatiotemporal kriging. June 2020.

[80] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.*, 32(1):4–24, January 2021.

[81] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. February 2020.

[82] Jixing Xu, Zhenlong Zhu, Jianxin Zhao, Xuanye Liu, Minghui Shan, and Jiecheng Guo. Gemini: A novel and universal heterogeneous graph information fusing framework for online recommendations. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 3356–3365, New York, NY, USA, August 2020. Association for Computing Machinery.

[83] Gui-Rong Xue, Chenxi Lin, Qiang Yang, Wensi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 114–121, New York, NY, USA, August 2005. Association for Computing Machinery.

[84] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for Skeleton-Based action recognition. January 2018.

[85] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for Web-Scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 974–983, New York, NY, USA, July 2018. Association for Computing Machinery.

[86] YouTube. YouTube for press. https://blog.youtube/press/. Accessed: 2022-10-3.

[87] Bo Zhang, Na Wang, and Hongxia Jin. Privacy concerns in online recommender systems: influences of control and user data input. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 159–173, 2014.

[88] Jiani Zhang, Xingjian Shi, Shenglin Zhao, and Irwin King. STAR-GCN: Stacked and reconstructed graph convolutional networks for recommender systems. May 2019.

[89] 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*, RecSys '18, pages 311–319, New York, NY, USA, September 2018. Association for Computing Machinery.

A
Rating evaluations

A.1. Jester

		Metric		
Item category	**Baseline**	**RMSE (95% CI)**	**MSE (95% CI)**	**MAE (95% CI)**
All	**Mean rating**	**4.0975 [4.0714 - 4.1230]**	**16.7894 [16.5765 - 16.9993]**	**3.1403 [3.1197 - 3.1606]**
	SVD++	4.2798 [4.2658 - 4.2940]	18.3170 [18.1968 - 18.4388]	3.3393 [3.3280 - 3.3506]
Low	Mean rating	5.0231 [4.4339 - 5.5994]	25.3200 [19.6593 - 31.3529]	4.0627 [3.4916 - 4.6352]
	SVD++	**4.2373 [3.7294 - 4.7418]**	**18.0225 [13.9087 - 22.4846]**	**3.4228 [2.9527 - 3.9098]**
Medium	**Mean rating**	**4.1592 [3.3458 - 5.0179]**	**17.4852 [11.1942 - 25.1794]**	**3.1554 [2.5603 - 3.8026]**
	SVD++	4.1777 [3.3298 - 5.0718]	17.6529 [11.0875 - 25.7234]	3.1431 [2.5439 - 3.7831]
High	**Mean rating**	**4.0957 [4.0699 - 4.1215]**	**16.7747 [16.5644 - 16.9870]**	**3.1387 [3.1184 - 3.1593]**
	SVD++	4.2799 [4.2659 - 4.2938]	18.3173 [18.1975 - 18.4367]	3.3393 [3.3281 - 3.3505]

Table A.1: Evaluations of the non-graph baselines for all items in the Jester dataset. Bold font indicates the best result for each category of items. Lower is better for all metrics.

			Metric		
Item cat.	**Conv. type**	**Attachment**	**RMSE (95% CI)**	**MSE (95% CI)**	**MAE (95% CI)**
All	kNN	True	3.3334 [3.3108 - 3.3562]	11.1119 [10.9614 - 11.2643]	2.5143 [2.4973 - 2.5313]
		BA	4.2766 [4.2494 - 4.3038]	18.2893 [18.0572 - 18.5226]	3.2699 [3.2495 - 3.2908]
		ER	4.1904 [4.1632 - 4.2174]	17.5600 [17.3324 - 17.7865]	3.2071 [3.1861 - 3.2281]
		Learned	**4.0835 [4.0582 - 4.1092]**	**16.6750 [16.4694 - 16.8858]**	**3.1281 [3.1078 - 3.1486]**
	GF	True	2.7818 [2.7623 - 2.8015]	7.7384 [7.6302 - 7.8486]	2.0888 [2.0747 - 2.1033]
		BA	4.3207 [4.2946 - 4.3476]	18.6689 [18.4437 - 18.9016]	3.3254 [3.3041 - 3.3470]
		ER	4.1101 [4.0846 - 4.1361]	16.8928 [16.6841 - 17.1075]	3.1594 [3.1392 - 3.1798]
		Learned	4.0903 [4.0644 - 4.1166]	16.7310 [16.5191 - 16.9466]	3.1341 [3.1139 - 3.1546]
	GCNN	True	4.0862 [4.0608 - 4.1115]	16.6971 [16.4904 - 16.9042]	3.1490 [3.1289 - 3.1690]
		BA	4.1047 [4.0801 - 4.1293]	16.8489 [16.6474 - 17.0512]	3.1712 [3.1515 - 3.1913]
		ER	4.1007 [4.0751 - 4.1270]	16.8161 [16.6066 - 17.0322]	3.1413 [3.1218 - 3.1618]
		Learned	4.0889 [4.0631 - 4.1147]	16.7194 [16.5090 - 16.9308]	3.1327 [3.1125 - 3.1526]
Low	kNN	True	5.1265 [4.5340 - 5.7321]	26.3736 [20.5571 - 32.8569]	4.1240 [3.5560 - 4.7242]
		BA	5.1976 [4.5867 - 5.7960]	27.1104 [21.0381 - 33.5934]	4.1925 [3.6027 - 4.8015]
		ER	5.1934 [4.5906 - 5.7825]	27.0651 [21.0738 - 33.4370]	4.1889 [3.6044 - 4.7814]
		Learned	4.3835 [3.8450 - 4.9263]	19.2893 [14.7841 - 24.2682]	3.5454 [3.0439 - 4.0625]
	GF	True	5.4810 [4.8295 - 6.1232]	30.1511 [23.3242 - 37.4937]	4.3855 [3.7548 - 5.0272]
		BA	4.3985 [3.8555 - 4.9419]	19.4231 [14.8646 - 24.4222]	3.5487 [3.0565 - 4.0558]
		ER	4.6523 [4.1331 - 5.1537]	21.7122 [17.0823 - 26.5610]	3.8151 [3.3044 - 4.3327]
		Learned	4.3056 [3.8360 - 4.7679]	18.5937 [14.7147 - 22.7326]	3.5478 [3.0739 - 4.0210]
	GCNN	True	4.2547 [3.7952 - 4.7010]	18.1555 [14.4035 - 22.0999]	3.5218 [3.0593 - 3.9851]
		BA	4.4872 [3.9977 - 4.9757]	20.1969 [15.9815 - 24.7575]	3.7192 [3.2388 - 4.2154]
		ER	4.2973 [3.8296 - 4.7573]	18.5232 [14.6657 - 22.6320]	3.5434 [3.0776 - 4.0123]
		Learned	**4.2727 [3.8221 - 4.7254]**	**18.3096 [14.6085 - 22.3290]**	**3.5237 [3.0708 - 3.9945]**
Medium	kNN	True	4.0159 [3.1822 - 4.8947]	16.3177 [10.1263 - 23.9582]	2.9785 [2.3879 - 3.6289]
		BA	**4.1512 [3.3470 - 5.0155]**	**17.4162 [11.2027 - 25.1548]**	**3.1505 [2.5577 - 3.8144]**
		ER	4.1667 [3.3577 - 5.0326]	17.5449 [11.2742 - 25.3266]	3.1583 [2.5674 - 3.8048]
		Learned	4.2759 [3.5140 - 5.0777]	18.4458 [12.3485 - 25.7831]	3.3048 [2.7001 - 3.9539]
	GF	True	4.2002 [3.3648 - 5.0791]	17.8337 [11.3220 - 25.7976]	3.1379 [2.5223 - 3.8209]
		BA	4.2762 [3.5106 - 5.0889]	18.4475 [12.3245 - 25.8964]	3.3093 [2.6958 - 3.9722]
		ER	4.2363 [3.4647 - 5.0532]	18.1139 [12.0043 - 25.5349]	3.2618 [2.6691 - 3.9140]
		Learned	4.2224 [3.4273 - 5.0449]	17.9993 [11.7461 - 25.4507]	3.2338 [2.6341 - 3.8742]
	GCNN	True	4.3298 [3.5745 - 5.1225]	18.9034 [12.7770 - 26.2396]	3.3823 [2.7852 - 4.0243]
		BA	4.2848 [3.5123 - 5.0913]	18.5209 [12.3364 - 25.9209]	3.3195 [2.7146 - 3.9644]
		ER	4.2407 [3.4688 - 5.0649]	18.1473 [12.0324 - 25.6531]	3.2678 [2.6643 - 3.9203]
		Learned	4.2488 [3.4695 - 5.0593]	18.2182 [12.0371 - 25.5970]	3.2731 [2.6718 - 3.9181]
High	kNN	True	3.3285 [3.3063 - 3.3512]	11.0792 [10.9319 - 11.2308]	2.5109 [2.4942 - 2.5276]
		BA	4.2748 [4.2473 - 4.3027]	18.2745 [18.0393 - 18.5131]	3.2685 [3.2476 - 3.2900]
		ER	4.1886 [4.1617 - 4.2161]	17.5447 [17.3201 - 17.7756]	3.2057 [3.1850 - 3.2265]
		Learned	**4.0828 [4.0568 - 4.1084]**	**16.6691 [16.4580 - 16.8791]**	**3.1274 [3.1069 - 3.1470]**
	GF	True	2.7733 [2.7539 - 2.7927]	7.6914 [7.5839 - 7.7991]	2.0839 [2.0699 - 2.0979]
		BA	4.3204 [4.2945 - 4.3469]	18.6662 [18.4430 - 18.8955]	3.3249 [3.3040 - 3.3462]
		ER	4.1085 [4.0830 - 4.1341]	16.8799 [16.6711 - 17.0906]	3.1578 [3.1379 - 3.1780]
		Learned	4.0897 [4.0639 - 4.1151]	16.7258 [16.5150 - 16.9338]	3.1333 [3.1129 - 3.1539]
	GCNN	True	4.0856 [4.0600 - 4.1109]	16.6921 [16.4836 - 16.8997]	3.1481 [3.1282 - 3.1686]
		BA	4.1037 [4.0785 - 4.1289]	16.8404 [16.6341 - 17.0475]	3.1701 [3.1501 - 3.1900]
		ER	4.1001 [4.0749 - 4.1251]	16.8108 [16.6048 - 17.0166]	3.1404 [3.1208 - 3.1601]
		Learned	4.0886 [4.0632 - 4.1145]	16.7172 [16.5095 - 16.9288]	3.1321 [3.1122 - 3.1523]

Table A.2: Evaluations for graph-based experiments on the Jester dataset. Bold font marks the best result for each category of items with true attachment excluded. Lower is better attachment for all metrics. The attachment column indicates the type of attachment that the model is trained and tested on.

A.2. MOVIELENS-1M

		Metric		
Item category	**Baseline**	**RMSE (95% CI)**	**MSE (95% CI)**	**MAE (95% CI)**
All	Mean rating	0.9547 [0.9503 - 0.9592]	0.9115 [0.9031 - 0.9200]	0.7432 [0.7397 - 0.7467]
	SVD++	**0.9458 [0.9418 - 0.9497]**	**0.8945 [0.8871 - 0.9019]**	**0.7578 [0.7544 - 0.7611]**
Low	Mean rating	1.1456 [0.9402 - 1.3563]	1.3236 [0.8840 - 1.8394]	0.9159 [0.7310 - 1.1130]
	SVD++	**1.1342 [0.9255 - 1.3429]**	**1.2975 [0.8565 - 1.8033]**	**0.9210 [0.7469 - 1.1087]**
Medium	Mean rating	0.9705 [0.9633 - 0.9776]	0.9419 [0.9279 - 0.9558]	0.7679 [0.7620 - 0.7739]
	SVD++	**0.9631 [0.9562 - 0.9699]**	**0.9276 [0.9142 - 0.9406]**	**0.7717 [0.7657 - 0.7775]**
High	Mean rating	0.9467 [0.9411 - 0.9522]	0.8962 [0.8856 - 0.9067]	0.7308 [0.7266 - 0.7351]
	SVD++	**0.9369 [0.9323 - 0.9416]**	**0.8778 [0.8692 - 0.8865]**	**0.7508 [0.7469 - 0.7548]**

Table A.3: Evaluations of the non-graph baselines for all items in the MovieLens-1M dataset. Bold font indicates the best result for each category of items. Lower is better for all metrics.

			Metric		
Item cat.	**Conv. type**	**Attachment**	**RMSE (95% CI)**	**MSE (95% CI)**	**MAE (95% CI)**
All	kNN	True	0.8399 [0.8360 - 0.8439]	0.7055 [0.6989 - 0.7122]	0.6532 [0.6501 - 0.6563]
		BA	1.0181 [1.0134 - 1.0228]	1.0366 [1.0270 - 1.0460]	0.7895 [0.7859 - 0.7932]
		ER	0.9958 [0.9911 - 1.0005]	0.9916 [0.9822 - 1.0010]	0.7721 [0.7685 - 0.7758]
		Learned	0.9675 [0.9632 - 0.9717]	0.9360 [0.9278 - 0.9441]	0.7605 [0.7570 - 0.7639]
	GF	True	0.7033 [0.7000 - 0.7066]	0.4946 [0.4900 - 0.4993]	0.5486 [0.5460 - 0.5512]
		BA	0.9408 [0.9366 - 0.9450]	0.8851 [0.8773 - 0.8930]	0.7439 [0.7406 - 0.7473]
		ER	**0.9402 [0.9362 - 0.9444]**	**0.8841 [0.8765 - 0.8918]**	**0.7427 [0.7394 - 0.7460]**
		Learned	0.9474 [0.9434 - 0.9514]	0.8975 [0.8900 - 0.9052]	0.7539 [0.7507 - 0.7573]
	GCNN	True	0.6929 [0.6897 - 0.6961]	0.4801 [0.4757 - 0.4845]	0.5420 [0.5395 - 0.5446]
		BA	0.9463 [0.9424 - 0.9503]	0.8955 [0.8880 - 0.9030]	0.7580 [0.7546 - 0.7613]
		ER	0.9412 [0.9372 - 0.9453]	0.8858 [0.8784 - 0.8935]	0.7468 [0.7436 - 0.7502]
		Learned	0.9495 [0.9457 - 0.9533]	0.9015 [0.8943 - 0.9088]	0.7640 [0.7608 - 0.7673]
Low	kNN	True	0.9962 [0.8238 - 1.1651]	1.0001 [0.6786 - 1.3576]	0.8012 [0.6407 - 0.9719]
		BA	1.0886 [0.9091 - 1.2678]	1.1934 [0.8265 - 1.6073]	0.8874 [0.7213 - 1.0641]
		ER	1.2009 [0.9797 - 1.4215]	1.4550 [0.9598 - 2.0207]	0.9740 [0.7909 - 1.1702]
		Learned	1.2476 [0.9997 - 1.4930]	1.5721 [0.9994 - 2.2290]	0.9930 [0.7909 - 1.2159]
	GF	True	1.0698 [0.8516 - 1.3013]	1.1579 [0.7253 - 1.6933]	0.8210 [0.6351 - 1.0174]
		BA	**1.0645 [0.8888 - 1.2380]**	**1.1411 [0.7900 - 1.5326]**	**0.8767 [0.7150 - 1.0476]**
		ER	1.0972 [0.8981 - 1.3003]	1.2144 [0.8066 - 1.6907]	0.8908 [0.7185 - 1.0703]
		Learned	1.1238 [0.9015 - 1.3510]	1.2763 [0.8127 - 1.8252]	0.8895 [0.7014 - 1.0886]
	GCNN	True	1.0938 [0.8933 - 1.2888]	1.2065 [0.7980 - 1.6610]	0.8552 [0.6695 - 1.0471]
		BA	1.0911 [0.8918 - 1.2931]	1.2008 [0.7953 - 1.6722]	0.8919 [0.7186 - 1.0734]
		ER	1.0912 [0.8933 - 1.2953]	1.2013 [0.7980 - 1.6779]	0.8949 [0.7252 - 1.0743]
		Learned	1.0815 [0.8822 - 1.2818]	1.1801 [0.7783 - 1.6430]	0.8698 [0.6992 - 1.0518]
Medium	kNN	True	0.8745 [0.8680 - 0.8811]	0.7647 [0.7533 - 0.7763]	0.6899 [0.6846 - 0.6954]
		BA	0.9884 [0.9808 - 0.9959]	0.9769 [0.9621 - 0.9919]	0.7809 [0.7747 - 0.7870]
		ER	0.9863 [0.9790 - 0.9937]	0.9728 [0.9585 - 0.9874]	0.7800 [0.7741 - 0.7861]
		Learned	1.0363 [1.0288 - 1.0438]	1.0739 [1.0584 - 1.0894]	0.8197 [0.8133 - 0.8261]
	GF	True	0.7824 [0.7761 - 0.7886]	0.6121 [0.6024 - 0.6220]	0.6146 [0.6095 - 0.6195]
		BA	0.9652 [0.9583 - 0.9719]	0.9317 [0.9184 - 0.9446]	0.7731 [0.7672 - 0.7790]
		ER	**0.9649 [0.9581 - 0.9717]**	**0.9310 [0.9179 - 0.9442]**	**0.7719 [0.7661 - 0.7777]**
		Learned	0.9750 [0.9681 - 0.9820]	0.9506 [0.9372 - 0.9643]	0.7761 [0.7702 - 0.7821]
	GCNN	True	0.7671 [0.7611 - 0.7732]	0.5884 [0.5793 - 0.5978]	0.6042 [0.5994 - 0.6090]
		BA	0.9706 [0.9640 - 0.9772]	0.9421 [0.9294 - 0.9550]	0.7805 [0.7748 - 0.7863]
		ER	0.9666 [0.9597 - 0.9736]	0.9343 [0.9210 - 0.9479]	0.7722 [0.7662 - 0.7781]
		Learned	0.9763 [0.9697 - 0.9829]	0.9531 [0.9402 - 0.9660]	0.7870 [0.7812 - 0.7928]
High	kNN	True	0.8222 [0.8173 - 0.8271]	0.6761 [0.6681 - 0.6842]	0.6349 [0.6312 - 0.6387]
		BA	1.0324 [1.0260 - 1.0385]	1.0658 [1.0528 - 1.0785]	0.7937 [0.7889 - 0.7984]
		ER	1.0002 [0.9942 - 1.0062]	1.0004 [0.9884 - 1.0124]	0.7681 [0.7635 - 0.7726]
		Learned	0.9315 [0.9263 - 0.9368]	0.8677 [0.8580 - 0.8776]	0.7312 [0.7271 - 0.7354]
	GF	True	0.6605 [0.6567 - 0.6642]	0.4362 [0.4313 - 0.4412]	0.5159 [0.5129 - 0.5188]
		BA	0.9284 [0.9232 - 0.9335]	0.8619 [0.8523 - 0.8714]	0.7295 [0.7254 - 0.7336]
		ER	**0.9277 [0.9225 - 0.9330]**	**0.8607 [0.8510 - 0.8704]**	**0.7282 [0.7240 - 0.7323]**
		Learned	0.9334 [0.9285 - 0.9383]	0.8713 [0.8621 - 0.8805]	0.7430 [0.7389 - 0.7470]
	GCNN	True	0.6529 [0.6492 - 0.6565]	0.4262 [0.4215 - 0.4310]	0.5112 [0.5083 - 0.5141]
		BA	0.9339 [0.9291 - 0.9387]	0.8722 [0.8632 - 0.8811]	0.7467 [0.7426 - 0.7507]
		ER	0.9283 [0.9233 - 0.9333]	0.8617 [0.8525 - 0.8711]	0.7342 [0.7302 - 0.7383]
		Learned	0.9359 [0.9313 - 0.9406]	0.8760 [0.8674 - 0.8847]	0.7527 [0.7489 - 0.7566]

Table A.4: Evaluations for graph-based experiments on the MovieLens-1M dataset. Bold font marks the best result for each category of items with true attachment excluded. Lower is better attachment for All metrics. The attachment column indicates the type of attachment that the model is trained and tested on.

A.3. YELP

		Metric		
Item category	**Baseline**	**RMSE (95% CI)**	**MSE (95% CI)**	**MAE (95% CI)**
All	Mean rating	0.9525 [0.9456 - 0.9593]	0.9073 [0.8942 - 0.9203]	0.7368 [0.7315 - 0.7423]
	SVD++	**0.9327 [0.9261 - 0.9392]**	**0.8699 [0.8576 - 0.8822]**	**0.7223 [0.7173 - 0.7274]**
Low	Mean rating	0.9885 [0.9776 - 0.9991]	0.9771 [0.9557 - 0.9982]	0.7642 [0.7557 - 0.7728]
	SVD++	**0.9706 [0.9601 - 0.9811]**	**0.9422 [0.9218 - 0.9626]**	**0.7532 [0.7451 - 0.7613]**
Medium	Mean rating	0.9383 [0.9259 - 0.9508]	0.8804 [0.8572 - 0.9039]	0.7271 [0.7175 - 0.7368]
	SVD++	**0.9157 [0.9038 - 0.9273]**	**0.8385 [0.8168 - 0.8599]**	**0.7099 [0.7008 - 0.7189]**
High	Mean rating	0.9126 [0.8997 - 0.9254]	0.8328 [0.8095 - 0.8564]	0.7069 [0.6972 - 0.7166]
	SVD++	**0.8932 [0.8810 - 0.9054]**	**0.7979 [0.7762 - 0.8198]**	**0.6907 [0.6817 - 0.7001]**

Table A.5: Evaluations of the non-graph baselines for the Yelp dataset. Bold font indicates the best result for each category of items. Lower is better for all metrics.

			Metric		
Item cat.	**Conv. type**	**Attachment**	**RMSE (95% CI)**	**MSE (95% CI)**	**MAE (95% CI)**
All	kNN	True	0.7929 [0.7867 - 0.7991]	0.6287 [0.6190 - 0.6385]	0.6021 [0.5975 - 0.6066]
		BA	0.9271 [0.9173 - 0.9365]	0.8595 [0.8415 - 0.8771]	0.7163 [0.7089 - 0.7235]
		ER	0.9271 [0.9175 - 0.9368]	0.8596 [0.8418 - 0.8776]	0.7164 [0.7088 - 0.7238]
		Learned	0.9609 [0.9543 - 0.9676]	0.9233 [0.9106 - 0.9362]	0.7599 [0.7547 - 0.7651]
	GF	True	0.7426 [0.7366 - 0.7485]	0.5514 [0.5426 - 0.5603]	0.5611 [0.5569 - 0.5656]
		BA	0.9031 [0.8938 - 0.9125]	0.8155 [0.7989 - 0.8326]	0.7055 [0.6983 - 0.7127]
		ER	0.9034 [0.8941 - 0.9127]	0.8161 [0.7994 - 0.8330]	0.7046 [0.6975 - 0.7117]
		Learned	0.9226 [0.9157 - 0.9292]	0.8511 [0.8386 - 0.8635]	0.7193 [0.7142 - 0.7244]
	GCNN	True	0.6933 [0.6876 - 0.6990]	0.4807 [0.4729 - 0.4886]	0.5253 [0.5212 - 0.5294]
		BA	**0.9024 [0.8931 - 0.9116]**	**0.8144 [0.7975 - 0.8310]**	**0.7023 [0.6953 - 0.7094]**
		ER	0.9039 [0.8946 - 0.9132]	0.8171 [0.8003 - 0.8339]	0.7034 [0.6962 - 0.7105]
		Learned	0.9235 [0.9172 - 0.9300]	0.8528 [0.8412 - 0.8650]	0.7295 [0.7245 - 0.7347]
Low	kNN	True	0.8709 [0.8611 - 0.8807]	0.7585 [0.7415 - 0.7756]	0.6654 [0.6579 - 0.6730]
		BA	0.9720 [0.9487 - 0.9956]	0.9450 [0.9000 - 0.9912]	0.7494 [0.7309 - 0.7677]
		ER	0.9721 [0.9485 - 0.9962]	0.9452 [0.8996 - 0.9925]	0.7494 [0.7307 - 0.7682]
		Learned	1.0082 [0.9973 - 1.0191]	1.0165 [0.9946 - 1.0386]	0.7959 [0.7875 - 0.8045]
	GF	True	0.8097 [0.8002 - 0.8195]	0.6557 [0.6403 - 0.6716]	0.6150 [0.6078 - 0.6223]
		BA	**0.9472 [0.9244 - 0.9703]**	**0.8972 [0.8546 - 0.9414]**	**0.7363 [0.7186 - 0.7542]**
		ER	0.9498 [0.9263 - 0.9736]	0.9022 [0.8580 - 0.9480]	0.7366 [0.7188 - 0.7545]
		Learned	0.9524 [0.9420 - 0.9629]	0.9071 [0.8874 - 0.9271]	0.7395 [0.7314 - 0.7476]
	GCNN	True	0.7599 [0.7511 - 0.7691]	0.5775 [0.5641 - 0.5915]	0.5767 [0.5699 - 0.5835]
		BA	0.9483 [0.9251 - 0.9715]	0.8995 [0.8558 - 0.9439]	0.7349 [0.7171 - 0.7527]
		ER	0.9482 [0.9256 - 0.9715]	0.8992 [0.8567 - 0.9438]	0.7350 [0.7174 - 0.7532]
		Learned	0.9522 [0.9422 - 0.9623]	0.9068 [0.8878 - 0.9261]	0.7503 [0.7424 - 0.7584]
Medium	kNN	True	0.7820 [0.7707 - 0.7932]	0.6115 [0.5940 - 0.6291]	0.5975 [0.5891 - 0.6060]
		BA	0.9236 [0.9073 - 0.9398]	0.8531 [0.8232 - 0.8831]	0.7142 [0.7013 - 0.7271]
		ER	0.9235 [0.9073 - 0.9398]	0.8530 [0.8232 - 0.8832]	0.7142 [0.7016 - 0.7267]
		Learned	0.9418 [0.9297 - 0.9538]	0.8870 [0.8643 - 0.9098]	0.7452 [0.7358 - 0.7548]
	GF	True	0.7331 [0.7225 - 0.7438]	0.5375 [0.5221 - 0.5532]	0.5582 [0.5504 - 0.5660]
		BA	0.9002 [0.8844 - 0.9160]	0.8104 [0.7822 - 0.8391]	0.7030 [0.6909 - 0.7154]
		ER	0.9011 [0.8853 - 0.9168]	0.8120 [0.7838 - 0.8405]	0.7025 [0.6905 - 0.7148]
		Learned	0.9107 [0.8988 - 0.9227]	0.8295 [0.8079 - 0.8514]	0.7105 [0.7012 - 0.7197]
	GCNN	True	0.6836 [0.6737 - 0.6938]	0.4673 [0.4538 - 0.4813]	0.5232 [0.5160 - 0.5306]
		BA	**0.8990 [0.8828 - 0.9153]**	**0.8083 [0.7793 - 0.8377]**	**0.7001 [0.6876 - 0.7127]**
		ER	0.9002 [0.8843 - 0.9162]	0.8105 [0.7820 - 0.8394]	0.7008 [0.6884 - 0.7132]
		Learned	0.9118 [0.9004 - 0.9234]	0.8314 [0.8107 - 0.8528]	0.7210 [0.7119 - 0.7303]
High	kNN	True	0.6748 [0.6647 - 0.6850]	0.4554 [0.4418 - 0.4693]	0.5134 [0.5062 - 0.5206]
		BA	0.9127 [0.8987 - 0.9266]	0.8330 [0.8077 - 0.8586]	0.7058 [0.6953 - 0.7164]
		ER	0.9124 [0.8985 - 0.9265]	0.8324 [0.8073 - 0.8583]	0.7056 [0.6951 - 0.7161]
		Learned	0.9074 [0.8956 - 0.9194]	0.8235 [0.8022 - 0.8454]	0.7219 [0.7127 - 0.7311]
	GF	True	0.6416 [0.6316 - 0.6514]	0.4117 [0.3989 - 0.4243]	0.4847 [0.4776 - 0.4917]
		BA	0.8884 [0.8753 - 0.9016]	0.7892 [0.7662 - 0.8128]	0.6959 [0.6858 - 0.7058]
		ER	0.8876 [0.8746 - 0.9007]	0.7878 [0.7650 - 0.8113]	0.6946 [0.6848 - 0.7044]
		Learned	0.8896 [0.8775 - 0.9016]	0.7913 [0.7700 - 0.8129]	0.6987 [0.6895 - 0.7080]
	GCNN	True	0.5931 [0.5837 - 0.6025]	0.3518 [0.3407 - 0.3630]	0.4515 [0.4450 - 0.4580]
		BA	**0.8875 [0.8743 - 0.9012]**	**0.7877 [0.7644 - 0.8121]**	**0.6920 [0.6821 - 0.7023]**
		ER	0.8896 [0.8764 - 0.9030]	0.7914 [0.7681 - 0.8155]	0.6936 [0.6837 - 0.7036]
		Learned	0.8920 [0.8807 - 0.9038]	0.7956 [0.7756 - 0.8168]	0.7076 [0.6986 - 0.7167]

Table A.6: Evaluations for graph-based experiments on the Yelp dataset. Bold font marks the best result for each category of items with true attachment excluded. Lower is better attachment for all metrics. The attachment column indicates the type of attachment that the model is trained and tested on.

B

Ranking evaluations

B.1. JESTER

		Metric		
Item category	**Baseline**	**NDCG (95% CI)**	**Recall (95% CI)**	**mAP (95%)**
All	Mean rating	0.6631 [0.6561 - 0.6701]	**0.9479 [0.9435 - 0.9523]**	0.4151 [0.4101 - 0.4200]
	SVD++	**0.6852 [0.6816 - 0.6889]**	0.8169 [0.8115 - 0.8223]	**0.6461 [0.6424 - 0.6498]**
Low	Mean rating	**0.7786 [0.6996 - 0.8522]**	0.7932 [0.7011 - 0.8736]	0.1763 [0.1519 - 0.2008]
	SVD++	0.7551 [0.6732 - 0.8321]	**0.8047 [0.7241 - 0.8851]**	**0.1796 [0.1553 - 0.2041]**
Medium	Mean rating	0.6074 [0.5200 - 0.6880]	**0.5678 [0.4800 - 0.6560]**	**0.1062 [0.0885 - 0.1240]**
	SVD++	**0.6077 [0.5200 - 0.6960]**	0.5673 [0.4800 - 0.6480]	0.1061 [0.0883 - 0.1244]
High	Mean rating	0.6636 [0.6567 - 0.6706]	**0.9480 [0.9434 - 0.9524]**	0.4144 [0.4095 - 0.4194]
	SVD++	**0.6854 [0.6818 - 0.6889]**	0.8174 [0.8120 - 0.8227]	**0.6460 [0.6423 - 0.6497]**

Table B.1: Evaluations of non-graph ranking experiments on the Jester dataset. Bold font indicates the best result for each category of items for that metric. Higher is better for all metrics.

			Metric		
Item cat.	**Conv. type**	**Attachment**	**NDCG (95% CI)**	**Recall (95% CI)**	**mAP (95%)**
All	kNN	True	0.8419 [0.8367 - 0.8471]	0.9543 [0.9497 - 0.9586]	0.4619 [0.4564 - 0.4672]
		BA	0.6588 [0.6519 - 0.6657]	0.9474 [0.9429 - 0.9518]	0.4140 [0.4091 - 0.4190]
		ER	0.6587 [0.6516 - 0.6659]	0.9476 [0.9430 - 0.9520]	0.4142 [0.4092 - 0.4193]
		Learned	0.6494 [0.6424 - 0.6564]	0.9469 [0.9424 - 0.9512]	0.4118 [0.4069 - 0.4167]
	GF	True	0.8973 [0.8930 - 0.9016]	0.9595 [0.9551 - 0.9638]	0.4790 [0.4735 - 0.4848]
		BA	0.6565 [0.6494 - 0.6636]	0.9474 [0.9429 - 0.9518]	0.4140 [0.4090 - 0.4191]
		ER	0.6559 [0.6487 - 0.6630]	0.9476 [0.9430 - 0.9520]	0.4140 [0.4090 - 0.4188]
		Learned	**0.6631 [0.6561 - 0.6701]**	**0.9478 [0.9433 - 0.9521]**	**0.4149 [0.4100 - 0.4198]**
	GCNN	True	0.9024 [0.8982 - 0.9065]	0.9597 [0.9552 - 0.9640]	0.4799 [0.4741 - 0.4855]
		BA	0.6558 [0.6488 - 0.6630]	0.9475 [0.9430 - 0.9518]	0.4140 [0.4091 - 0.4189]
		ER	0.6559 [0.6488 - 0.6630]	0.9475 [0.9431 - 0.9519]	0.4140 [0.4090 - 0.4189]
		Learned	0.6595 [0.6525 - 0.6666]	0.9475 [0.9430 - 0.9518]	0.4144 [0.4095 - 0.4195]
Low	kNN	True	0.7672 [0.6891 - 0.8402]	0.7928 [0.7011 - 0.8736]	0.1762 [0.1506 - 0.2010]
		BA	0.7782 [0.6988 - 0.8510]	**0.7935 [0.7011 - 0.8736]**	**0.1765 [0.1513 - 0.2019]**
		ER	**0.7783 [0.6987 - 0.8523]**	0.7932 [0.7011 - 0.8736]	0.1763 [0.1518 - 0.2013]
		Learned	0.6792 [0.5903 - 0.7665]	0.7931 [0.7011 - 0.8736]	0.1713 [0.1471 - 0.1962]
	GF	True	0.7863 [0.7075 - 0.8598]	0.7941 [0.7011 - 0.8736]	0.1779 [0.1534 - 0.2030]
		BA	0.7422 [0.6597 - 0.8218]	0.7930 [0.7011 - 0.8736]	0.1752 [0.1509 - 0.2003]
		ER	0.7781 [0.6987 - 0.8513]	0.7924 [0.7011 - 0.8736]	0.1760 [0.1518 - 0.2007]
		Learned	0.7422 [0.6593 - 0.8191]	0.7924 [0.7011 - 0.8736]	0.1750 [0.1504 - 0.1997]
	GCNN	True	0.7819 [0.7023 - 0.8552]	0.7929 [0.7011 - 0.8736]	0.1774 [0.1531 - 0.2023]
		BA	0.7417 [0.6577 - 0.8202]	0.7926 [0.7011 - 0.8736]	0.1751 [0.1508 - 0.2002]
		ER	0.7414 [0.6573 - 0.8204]	0.7941 [0.7011 - 0.8736]	0.1755 [0.1509 - 0.2003]
		Learned	0.7417 [0.6568 - 0.8211]	0.7932 [0.7011 - 0.8736]	0.1752 [0.1505 - 0.1995]
Medium	kNN	True	0.6560 [0.5680 - 0.7360]	0.5681 [0.4800 - 0.6560]	0.1083 [0.0899 - 0.1268]
		BA	0.6082 [0.5200 - 0.6960]	0.5681 [0.4800 - 0.6560]	0.1062 [0.0885 - 0.1246]
		ER	0.6088 [0.5200 - 0.6960]	0.5681 [0.4800 - 0.6560]	0.1063 [0.0885 - 0.1244]
		Learned	0.6085 [0.5200 - 0.6960]	0.5679 [0.4800 - 0.6560]	0.1062 [0.0883 - 0.1244]
	GF	True	0.7278 [0.6480 - 0.8000]	0.5683 [0.4800 - 0.6560]	0.1099 [0.0911 - 0.1282]
		BA	0.6085 [0.5200 - 0.6880]	0.5679 [0.4800 - 0.6560]	0.1062 [0.0881 - 0.1242]
		ER	**0.6091 [0.5200 - 0.6960]**	0.5681 [0.4800 - 0.6560]	0.1063 [0.0879 - 0.1244]
		Learned	0.6085 [0.5200 - 0.6880]	**0.5684 [0.4800 - 0.6560]**	0.1063 [0.0881 - 0.1248]
	GCNN	True	0.7281 [0.6480 - 0.8080]	0.5678 [0.4800 - 0.6560]	0.1098 [0.0918 - 0.1284]
		BA	0.6084 [0.5200 - 0.6960]	0.5685 [0.4800 - 0.6560]	**0.1063 [0.0885 - 0.1246]**
		ER	0.6078 [0.5200 - 0.6960]	0.5675 [0.4800 - 0.6560]	0.1061 [0.0887 - 0.1246]
		Learned	0.6080 [0.5200 - 0.6882]	0.5678 [0.4800 - 0.6560]	0.1062 [0.0887 - 0.1242]
High	kNN	True	0.8429 [0.8376 - 0.8480]	0.9543 [0.9498 - 0.9587]	0.4610 [0.4555 - 0.4666]
		BA	0.6594 [0.6525 - 0.6663]	0.9476 [0.9430 - 0.9520]	0.4134 [0.4084 - 0.4184]
		ER	0.6592 [0.6522 - 0.6662]	0.9478 [0.9432 - 0.9522]	0.4135 [0.4086 - 0.4185]
		Learned	0.6503 [0.6431 - 0.6575]	0.9473 [0.9428 - 0.9516]	0.4111 [0.4060 - 0.4161]
	GF	True	0.8982 [0.8938 - 0.9025]	0.9595 [0.9551 - 0.9638]	0.4781 [0.4723 - 0.4838]
		BA	0.6574 [0.6503 - 0.6643]	0.9476 [0.9431 - 0.9520]	0.4134 [0.4084 - 0.4184]
		ER	0.6565 [0.6495 - 0.6636]	0.9477 [0.9431 - 0.9522]	0.4132 [0.4081 - 0.4182]
		Learned	**0.6636 [0.6566 - 0.6707]**	**0.9479 [0.9434 - 0.9523]**	**0.4142 [0.4091 - 0.4191]**
	GCNN	True	0.9033 [0.8991 - 0.9074]	0.9597 [0.9552 - 0.9640]	0.4788 [0.4731 - 0.4846]
		BA	0.6564 [0.6493 - 0.6633]	0.9476 [0.9431 - 0.9519]	0.4132 [0.4083 - 0.4182]
		ER	0.6563 [0.6491 - 0.6633]	0.9476 [0.9431 - 0.9521]	0.4132 [0.4082 - 0.4182]
		Learned	0.6599 [0.6528 - 0.6670]	0.9477 [0.9431 - 0.9521]	0.4136 [0.4085 - 0.4185]

Table B.2: Evaluations of graph-based ranking experiments on the Jester dataset. Bold font marks the best result for each category of items with true attachment excluded. Higher is better attachment for all metrics. The attachment column indicates the type of attachment that the model is trained and tested on.

B.2. MOVIELENS-1M

		Metric		
Item category	**Baseline**	**NDCG (95% CI)**	**Recall (95% CI)**	**mAP (95%)**
All	Mean rating	0.7485 [0.7413 - 0.7556]	**0.6113 [0.5961 - 0.6267]**	0.7258 [0.7168 - 0.7345]
	SVD++	**0.7535 [0.7469 - 0.7601]**	0.6099 [0.5946 - 0.6249]	**0.7267 [0.7178 - 0.7354]**
Low	Mean rating	0.8429 [0.8205 - 0.8652]	**0.5553 [0.5194 - 0.5908]**	0.1324 [0.1217 - 0.1434]
	SVD++	**0.8652 [0.8442 - 0.8853]**	0.5552 [0.5187 - 0.5907]	**0.1331 [0.1222 - 0.1442]**
Medium	Mean rating	0.7413 [0.7296 - 0.7529]	**0.7982 [0.7823 - 0.8138]**	**0.4388 [0.4255 - 0.4517]**
	SVD++	**0.7468 [0.7353 - 0.7582]**	0.7954 [0.7794 - 0.8109]	0.4374 [0.4246 - 0.4501]
High	Mean rating	0.7288 [0.7208 - 0.7367]	**0.6951 [0.6808 - 0.7094]**	**0.7072 [0.6979 - 0.7168]**
	SVD++	**0.7321 [0.7245 - 0.7396]**	0.6932 [0.6790 - 0.7076]	0.7060 [0.6964 - 0.7152]

Table B.3: Evaluations of the non-graph baselines for the MovieLens-1M dataset. Bold font indicates the best result for each category of items. Lower is better for all metrics.

			Metric		
Item cat.	**Conv. type**	**Attachment**	**NDCG (95% CI)**	**Recall (95% CI)**	**mAP (95%)**
All	kNN	True	0.8998 [0.8966 - 0.9030]	0.6602 [0.6455 - 0.6755]	0.8477 [0.8405 - 0.8546]
		BA	0.7410 [0.7339 - 0.7480]	0.6080 [0.5927 - 0.6232]	0.7175 [0.7088 - 0.7263]
		ER	0.7443 [0.7371 - 0.7513]	0.6087 [0.5933 - 0.6240]	0.7198 [0.7110 - 0.7284]
		Learned	0.7326 [0.7257 - 0.7394]	0.6040 [0.5887 - 0.6193]	0.7095 [0.7008 - 0.7182]
	GF	True	0.9441 [0.9418 - 0.9465]	0.6850 [0.6698 - 0.7000]	0.8840 [0.8772 - 0.8907]
		BA	0.7419 [0.7346 - 0.7490]	0.6093 [0.5942 - 0.6243]	0.7172 [0.7084 - 0.7258]
		ER	0.7424 [0.7352 - 0.7495]	0.6096 [0.5946 - 0.6249]	0.7182 [0.7093 - 0.7270]
		Learned	**0.7486 [0.7417 - 0.7555]**	0.6110 [0.5959 - 0.6262]	0.7251 [0.7163 - 0.7339]
	GCNN	True	0.9462 [0.9438 - 0.9486]	0.6870 [0.6719 - 0.7020]	0.8862 [0.8792 - 0.8929]
		BA	0.7424 [0.7352 - 0.7496]	0.6087 [0.5933 - 0.6241]	0.7186 [0.7097 - 0.7274]
		ER	0.7429 [0.7357 - 0.7501]	0.6088 [0.5936 - 0.6240]	0.7188 [0.7101 - 0.7274]
		Learned	0.7485 [0.7414 - 0.7554]	**0.6118 [0.5967 - 0.6272]**	**0.7254 [0.7166 - 0.7344]**
Low	kNN	True	0.8683 [0.8472 - 0.8882]	0.5553 [0.5188 - 0.5908]	0.1350 [0.1236 - 0.1465]
		BA	0.8416 [0.8187 - 0.8636]	0.5553 [0.5190 - 0.5915]	0.1324 [0.1213 - 0.1436]
		ER	0.8434 [0.8204 - 0.8650]	**0.5556 [0.5201 - 0.5908]**	0.1326 [0.1216 - 0.1437]
		Learned	0.8178 [0.7942 - 0.8414]	0.5547 [0.5186 - 0.5894]	0.1303 [0.1192 - 0.1413]
	GF	True	0.8713 [0.8506 - 0.8916]	0.5556 [0.5189 - 0.5908]	0.1358 [0.1245 - 0.1475]
		BA	0.8401 [0.8172 - 0.8618]	0.5553 [0.5190 - 0.5908]	**0.1328 [0.1219 - 0.1440]**
		ER	0.8373 [0.8142 - 0.8591]	0.5553 [0.5197 - 0.5912]	0.1325 [0.1217 - 0.1437]
		Learned	**0.8448 [0.8220 - 0.8667]**	0.5553 [0.5199 - 0.5908]	0.1326 [0.1217 - 0.1437]
	GCNN	True	0.8738 [0.8534 - 0.8933]	0.5553 [0.5190 - 0.5908]	0.1352 [0.1239 - 0.1469]
		BA	0.8422 [0.8203 - 0.8638]	0.5553 [0.5190 - 0.5908]	0.1321 [0.1210 - 0.1434]
		ER	0.8423 [0.8196 - 0.8648]	0.5555 [0.5190 - 0.5915]	0.1322 [0.1212 - 0.1433]
		Learned	0.8439 [0.8214 - 0.8658]	0.5554 [0.5190 - 0.5908]	0.1326 [0.1216 - 0.1438]
Medium	kNN	True	0.8527 [0.8443 - 0.8608]	0.8112 [0.7958 - 0.8264]	0.4824 [0.4688 - 0.4961]
		BA	0.7378 [0.7259 - 0.7495]	0.7983 [0.7828 - 0.8137]	0.4367 [0.4237 - 0.4495]
		ER	0.7370 [0.7250 - 0.7488]	0.7982 [0.7827 - 0.8136]	0.4376 [0.4251 - 0.4505]
		Learned	0.7184 [0.7063 - 0.7304]	0.7924 [0.7766 - 0.8085]	0.4249 [0.4125 - 0.4376]
	GF	True	0.9142 [0.9082 - 0.9200]	0.8226 [0.8072 - 0.8376]	0.5087 [0.4944 - 0.5226]
		BA	0.7389 [0.7271 - 0.7506]	**0.7983 [0.7824 - 0.8138]**	0.4368 [0.4242 - 0.4495]
		ER	0.7380 [0.7259 - 0.7496]	0.7980 [0.7822 - 0.8135]	0.4369 [0.4239 - 0.4495]
		Learned	0.7400 [0.7282 - 0.7516]	0.7982 [0.7826 - 0.8140]	0.4378 [0.4246 - 0.4509]
	GCNN	True	0.9185 [0.9126 - 0.9242]	0.8237 [0.8085 - 0.8386]	0.5113 [0.4971 - 0.5256]
		BA	0.7367 [0.7249 - 0.7483]	0.7977 [0.7818 - 0.8133]	0.4354 [0.4225 - 0.4482]
		ER	0.7376 [0.7260 - 0.7494]	0.7978 [0.7820 - 0.8134]	0.4363 [0.4235 - 0.4493]
		Learned	**0.7425 [0.7308 - 0.7536]**	0.7983 [0.7821 - 0.8141]	**0.4391 [0.4260 - 0.4520]**
High	kNN	True	0.8998 [0.8961 - 0.9033]	0.7377 [0.7243 - 0.7511]	0.8156 [0.8073 - 0.8239]
		BA	0.7205 [0.7122 - 0.7285]	0.6922 [0.6781 - 0.7063]	0.6994 [0.6900 - 0.7086]
		ER	0.7239 [0.7155 - 0.7320]	0.6928 [0.6784 - 0.7072]	0.7015 [0.6919 - 0.7111]
		Learned	0.7129 [0.7048 - 0.7209]	0.6893 [0.6752 - 0.7035]	0.6931 [0.6836 - 0.7025]
	GF	True	0.9456 [0.9431 - 0.9482]	0.7583 [0.7448 - 0.7717]	0.8485 [0.8398 - 0.8567]
		BA	0.7208 [0.7125 - 0.7290]	0.6931 [0.6785 - 0.7073]	0.6991 [0.6897 - 0.7085]
		ER	0.7221 [0.7136 - 0.7304]	0.6937 [0.6797 - 0.7077]	0.7002 [0.6907 - 0.7098]
		Learned	**0.7290 [0.7209 - 0.7370]**	**0.6946 [0.6803 - 0.7089]**	0.7060 [0.6965 - 0.7155]
	GCNN	True	0.9491 [0.9466 - 0.9516]	0.7606 [0.7472 - 0.7741]	0.8511 [0.8427 - 0.8594]
		BA	0.7210 [0.7126 - 0.7292]	0.6920 [0.6772 - 0.7065]	0.7003 [0.6907 - 0.7097]
		ER	0.7219 [0.7135 - 0.7302]	0.6924 [0.6785 - 0.7064]	0.7005 [0.6911 - 0.7096]
		Learned	0.7275 [0.7193 - 0.7355]	0.6944 [0.6805 - 0.7084]	**0.7065 [0.6969 - 0.7160]**

Table B.4: Evaluations of graph-based ranking experiments on the MovieLens-1M dataset. Bold font marks the best result for each category of items with true attachment excluded. Higher is better attachment for all metrics. The attachment column indicates the type of attachment that the model is trained and tested on.

B.3. Yelp

		Metric		
Item category	**Baseline**	**NDCG (95% CI)**	**Recall (95% CI)**	**mAP (95%)**
All	Mean rating	**0.7325 [0.7253 - 0.7397]**	0.9761 [0.9733 - 0.9788]	0.4789 [0.4731 - 0.4845]
	SVD++	0.7021 [0.6944 - 0.7098]	**0.9769 [0.9744 - 0.9793]**	**0.4832 [0.4775 - 0.4889]**
Low	Mean rating	**0.7945 [0.7852 - 0.8035]**	0.8842 [0.8746 - 0.8938]	**0.2956 [0.2899 - 0.3012]**
	SVD++	0.7554 [0.7457 - 0.7651]	**0.8875 [0.8779 - 0.8968]**	0.2947 [0.2892 - 0.3004]
Medium	Mean rating	**0.7910 [0.7807 - 0.8011]**	0.8529 [0.8417 - 0.8640]	0.2558 [0.2503 - 0.2612]
	SVD++	0.7875 [0.7776 - 0.7972]	**0.8611 [0.8507 - 0.8717]**	**0.2636 [0.2583 - 0.2690]**
High	Mean rating	0.7795 [0.7684 - 0.7904]	0.8620 [0.8508 - 0.8732]	0.2651 [0.2593 - 0.2711]
	SVD++	**0.7823 [0.7716 - 0.7928]**	**0.8643 [0.8536 - 0.8751]**	**0.2700 [0.2641 - 0.2759]**

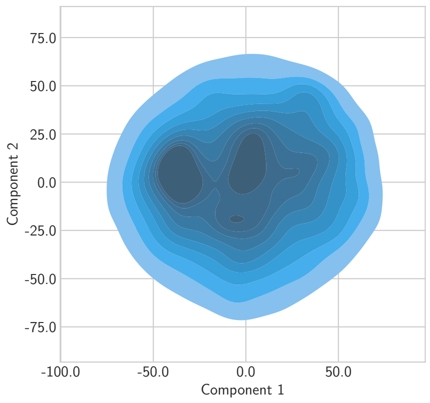
Table B.5: Evaluations of the non-graph baselines for the Yelp dataset. Bold font indicates the best result for each category of items. Lower is better for all metrics.

			Metric		
Item cat.	**Conv. type**	**Attachment**	**NDCG (95% CI)**	**Recall (95% CI)**	**mAP (95%)**
All	kNN	True	0.9118 [0.9083 - 0.9153]	0.9846 [0.9823 - 0.9869]	0.5349 [0.5288 - 0.5410]
		BA	0.7352 [0.7281 - 0.7425]	0.9763 [0.9735 - 0.9790]	0.4794 [0.4736 - 0.4851]
		ER	0.7346 [0.7276 - 0.7416]	0.9763 [0.9735 - 0.9790]	0.4791 [0.4734 - 0.4848]
		Learned	0.6193 [0.6109 - 0.6278]	0.9695 [0.9662 - 0.9727]	0.4446 [0.4391 - 0.4501]
	GF	True	0.9212 [0.9179 - 0.9245]	0.9853 [0.9830 - 0.9875]	0.5395 [0.5331 - 0.5458]
		BA	0.7355 [0.7282 - 0.7428]	0.9763 [0.9735 - 0.9791]	0.4794 [0.4736 - 0.4851]
		ER	0.7355 [0.7282 - 0.7427]	0.9762 [0.9734 - 0.9789]	0.4795 [0.4736 - 0.4853]
		Learned	0.7346 [0.7272 - 0.7418]	0.9760 [0.9732 - 0.9788]	0.4793 [0.4735 - 0.4850]
	GCNN	True	0.9324 [0.9294 - 0.9353]	0.9854 [0.9831 - 0.9876]	0.5429 [0.5367 - 0.5493]
		BA	0.7369 [0.7298 - 0.7441]	0.9762 [0.9733 - 0.9789]	0.4800 [0.4742 - 0.4857]
		ER	**0.7371 [0.7298 - 0.7443]**	**0.9762 [0.9733 - 0.9789]**	**0.4801 [0.4744 - 0.4857]**
		Learned	0.7346 [0.7273 - 0.7418]	0.9762 [0.9733 - 0.9790]	0.4798 [0.4740 - 0.4856]
Low	kNN	True	0.9011 [0.8949 - 0.9071]	0.8847 [0.8750 - 0.8941]	0.3118 [0.3060 - 0.3176]
		BA	0.7973 [0.7884 - 0.8062]	0.8842 [0.8746 - 0.8935]	0.2958 [0.2902 - 0.3014]
		ER	0.7974 [0.7882 - 0.8062]	0.8842 [0.8748 - 0.8937]	0.2959 [0.2904 - 0.3015]
		Learned	0.6801 [0.6694 - 0.6909]	0.8837 [0.8741 - 0.8930]	0.2772 [0.2719 - 0.2825]
	GF	True	0.9111 [0.9053 - 0.9168]	0.8847 [0.8750 - 0.8941]	0.3135 [0.3076 - 0.3196]
		BA	0.7984 [0.7896 - 0.8073]	0.8842 [0.8747 - 0.8938]	**0.2961 [0.2906 - 0.3018]**
		ER	**0.7987 [0.7898 - 0.8075]**	**0.8843 [0.8744 - 0.8940]**	0.2961 [0.2903 - 0.3019]
		Learned	0.7952 [0.7862 - 0.8042]	0.8841 [0.8742 - 0.8939]	0.2957 [0.2900 - 0.3014]
	GCNN	True	0.9186 [0.9131 - 0.9239]	0.8846 [0.8752 - 0.8939]	0.3146 [0.3088 - 0.3206]
		BA	0.7985 [0.7895 - 0.8071]	0.8842 [0.8748 - 0.8939]	0.2962 [0.2906 - 0.3018]
		ER	0.7986 [0.7897 - 0.8075]	0.8842 [0.8746 - 0.8937]	0.2962 [0.2905 - 0.3020]
		Learned	0.7986 [0.7897 - 0.8076]	0.8841 [0.8743 - 0.8936]	0.2960 [0.2904 - 0.3017]
Medium	kNN	True	0.9152 [0.9087 - 0.9216]	0.8529 [0.8415 - 0.8638]	0.2708 [0.2653 - 0.2764]
		BA	0.7940 [0.7837 - 0.8042]	0.8529 [0.8418 - 0.8639]	0.2561 [0.2507 - 0.2616]
		ER	0.7938 [0.7836 - 0.8039]	0.8530 [0.8418 - 0.8639]	0.2561 [0.2507 - 0.2615]
		Learned	0.7343 [0.7230 - 0.7454]	0.8529 [0.8417 - 0.8638]	0.2486 [0.2433 - 0.2539]
	GF	True	0.9229 [0.9167 - 0.9289]	0.8529 [0.8418 - 0.8636]	0.2719 [0.2661 - 0.2775]
		BA	0.7956 [0.7856 - 0.8058]	**0.8529 [0.8418 - 0.8636]**	0.2561 [0.2508 - 0.2614]
		ER	0.7953 [0.7851 - 0.8054]	0.8529 [0.8418 - 0.8641]	0.2561 [0.2506 - 0.2615]
		Learned	**0.7962 [0.7860 - 0.8062]**	0.8529 [0.8417 - 0.8636]	0.2561 [0.2507 - 0.2615]
	GCNN	True	0.9312 [0.9256 - 0.9367]	0.8530 [0.8418 - 0.8641]	0.2724 [0.2666 - 0.2782]
		BA	0.7947 [0.7843 - 0.8049]	0.8529 [0.8415 - 0.8641]	0.2562 [0.2508 - 0.2617]
		ER	0.7955 [0.7853 - 0.8054]	0.8529 [0.8418 - 0.8639]	0.2561 [0.2507 - 0.2615]
		Learned	0.7956 [0.7855 - 0.8057]	0.8529 [0.8415 - 0.8640]	**0.2562 [0.2510 - 0.2617]**
High	kNN	True	0.9368 [0.9308 - 0.9426]	0.8622 [0.8510 - 0.8729]	0.2850 [0.2786 - 0.2913]
		BA	0.7788 [0.7678 - 0.7899]	0.8620 [0.8507 - 0.8730]	0.2652 [0.2594 - 0.2712]
		ER	0.7792 [0.7684 - 0.7901]	0.8621 [0.8510 - 0.8733]	0.2651 [0.2592 - 0.2710]
		Learned	0.7418 [0.7301 - 0.7534]	0.8616 [0.8503 - 0.8727]	0.2599 [0.2541 - 0.2656]
	GF	True	0.9469 [0.9413 - 0.9521]	0.8621 [0.8509 - 0.8731]	0.2864 [0.2801 - 0.2928]
		BA	0.7795 [0.7686 - 0.7903]	0.8620 [0.8509 - 0.8733]	0.2651 [0.2592 - 0.2711]
		ER	0.7802 [0.7694 - 0.7914]	0.8620 [0.8507 - 0.8729]	0.2653 [0.2595 - 0.2712]
		Learned	0.7822 [0.7712 - 0.7930]	**0.8621 [0.8508 - 0.8731]**	0.2654 [0.2596 - 0.2714]
	GCNN	True	0.9540 [0.9491 - 0.9588]	0.8622 [0.8511 - 0.8732]	0.2869 [0.2807 - 0.2933]
		BA	**0.7840 [0.7729 - 0.7947]**	0.8620 [0.8507 - 0.8731]	0.2655 [0.2597 - 0.2712]
		ER	0.7836 [0.7726 - 0.7944]	0.8619 [0.8506 - 0.8729]	0.2655 [0.2597 - 0.2713]
		Learned	0.7837 [0.7728 - 0.7945]	0.8620 [0.8506 - 0.8728]	**0.2657 [0.2599 - 0.2716]**

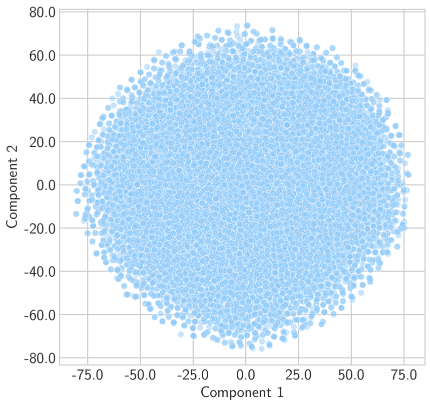
Table B.6: Evaluations of graph-based ranking experiments on the Yelp dataset. Bold font marks the best result for each category of items with true attachment excluded. Higher is better attachment for all metrics. The attachment column indicates the type of attachment that the model is trained and tested on.

C
FIGURES

C.1. JESTER



(a) KDE plot of t-SNE embedding



(b) Scatter plot of t-SNE embedding

Figure C.1: Visualization of the two-dimensional embedding of the latent factors of Jester users learned by the SVD++ algorithm. The embedding is calculated using t-SNE with a perplexity of 40.

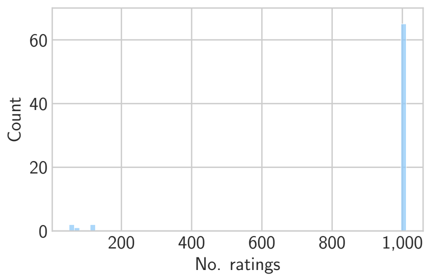


Figure C.2: Histogram of number of ratings for the Jester dataset

C.2. MovieLens-1M

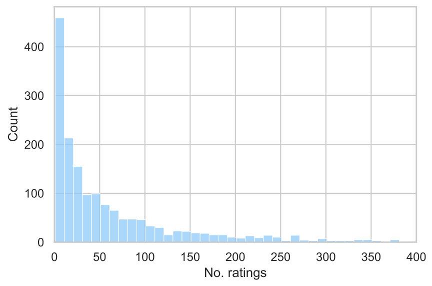
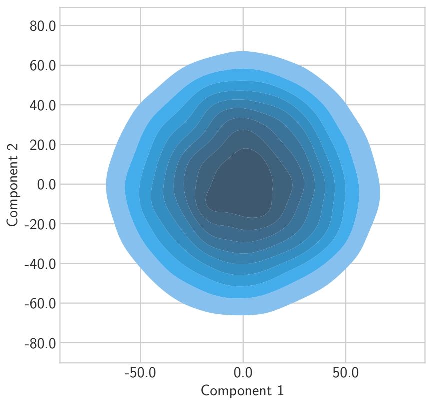
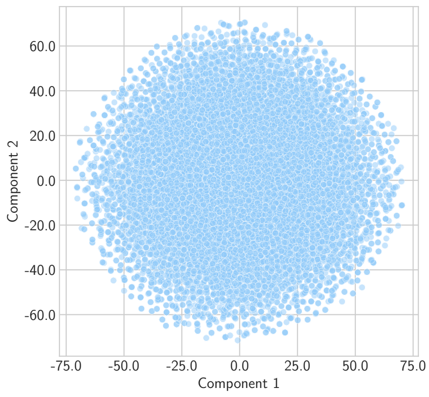


Figure C.3: Histogram of number of ratings for the MovieLens-1M dataset

C.3. YELP



(a) KDE plot of t-SNE embedding



(b) Scatter plot of t-SNE embedding

Figure C.4: Visualization of the two-dimensional embedding of the latent factors of Yelp users learned by the SVD++ algorithm. The embedding is calculated using t-SNE with a perplexity of 40.

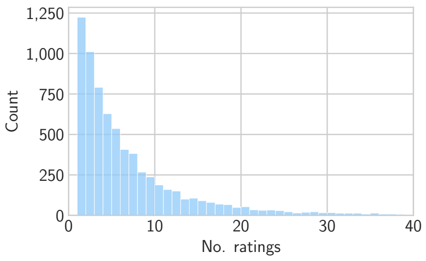


Figure C.5: Histogram of number of ratings for the Yelp dataset