Politecnico di Milano

School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

# Combining collaborative and content information into a novel graph based hybrid model for recommendation

Supervisor: Prof. Paolo Cremonesi
Co-Supervisor: Dott. Cesare Bernardis

Master thesis by:
Seydou Kane, 899519

Academic Year 2018-2019

# Abstract

Recommender systems are information filtering systems that are used to predict the preferences of users over a set of items in order to provide recommendations. Because of the extremely large number of products available in most online web services, recommender systems are extremely important to help the users navigate these huge catalogues. We encounter their usage every day. For instance, when *Spotify* create a personalized daily playlist for us, or when *Netflix* suggests us a new movie.
There are mainly two types of recommender systems: collaborative based and content based. The former uses the users' behaviors, their past interactions with the set of items, to provide recommendations, while the latter relies on the description of the items to find similar ones. In most cases, collaborative techniques have better performance. However, they are not able to recommend new items because of their lack of user interactions. Content based approaches, on the other hand, can include items that don't have any interaction yet in their recommendations. This type of situation, referred to as cold start, is very common in systems with a dynamic set of items.
In this thesis, we combine the two approaches into a single model, which uses a machine learning optimization algorithm to find the optimal balance, thus exploiting the advantages of both techniques.
The results of the conducted experiments indicate that the proposed model is able to outperform, in most cases, the state of art collaborative algorithms, while achieving results similar to those of content based

algorithms in cold start scenarios.

# Sommario

I sistemi di raccomandazione sono sistemi di filtraggio dei contenuti che vengono utilizzati per predire le preferenze degli utenti su un insieme di elementi al fine di fornire raccomandazioni. A causa del numero estremamente elevato di prodotti disponibili nella maggior parte delle applicazioni web, i sistemi di raccomandazione sono estremamente importanti per aiutare gli utenti a navigare in questi vasti cataloghi. Incontriamo il loro utilizzo tutti i giorni. Per esempio, quando *Spotify* crea una playlist giornaliera personalizzata per noi, o quando *Netflix* ci suggerisce un nuovo film. Ci sono principalmente due tipi di sistemi di raccomandazione: quelli collaborativi e quelli basati sui contenuti. Il primo utilizza i comportamenti degli utenti, le loro interazioni passate con l'insieme degli elementi, per fornire raccomandazioni, mentre il secondo si basa sulla descrizione degli elementi per trovarne di simili. Nella maggior parte dei casi, le tecniche collaborative hanno prestazioni migliori. Tuttavia, non sono in grado di raccomandare nuovi elementi a causa della loro mancanza di interazioni con gli utenti. Gli approcci basati sui contenuti, invece, possono includere nelle loro raccomandazioni elementi che non hanno ancora alcuna interazione. Questo tipo di scenario, chiamato *cold start*, è molto comune nei sistemi con un insieme dinamico di elementi.
In questa tesi, combiniamo i due approcci in un unico modello, che utilizza un algoritmo di machine learning di ottimizzazione per trovare l'equilibrio ottimale, sfruttando così i vantaggi di entrambe le tecniche. I risultati degli

esperimenti condotti indicano che il modello proposto è in grado di superare, nella maggior parte dei casi, gli algoritmi collaborativi dello stato dell'arte, e di ottenere risultati simili a quelli degli algoritmi basati sui contenuti in scenari di cold start .

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Over the past past few years, we observed an increasing use of the Web as a mean to provide services to users. Like with most businesses, the goal for these providers is to keep the customers using their services. The objective can be met only if users are able to find new items they are interested in. Here is where recommender systems come into play.
Recommender systems are information filtering systems that predict the preferences of users over a set of items in order to produce item recommendations. For example, for a service provider such as Netflix, it's crucial that its users continuously discover new shows they are interested in. They are also heavily used in social networks, in which also users are recommended to other users as friends, and in e-commerce services such as Amazon.
The most common recommender systems are mainly of two types:

- **Collaborative based**: they try to learn the taste of the users by analyzing their rating behaviors. In general, this type of recommendation performs better than the remaining ones, but, to get the best out of it, the target user needs to have enough ratings.

- **Content based**: they use items' descriptions to find items similar to the ones the target user has interacted with already. Normally, they are not able to provide recommendations as good as those produced by collaborative algorithms, but they can recommend new items, items that don't have any ratings yet.

As we've seen, both types have different strengths and weaknesses; this is the reason why they are often combined to create *hybrid* recommender systems. They are able to improve the general performance of the single recommenders, while exploiting the advantages of both types of systems.
In this thesis, we propose a new hybrid, graph based, machine learning model that uses collaborative and content information to combine the best of both approaches. In this new model, recommendations are based on three steps walks on the graph, starting from a user node and ending into an item node. We show that from a user node, there are two types of paths that can lead to an item node: the collaborative path and the content path. We unify these two kind of paths into a single one, and used Bayesian Personalized Ranking [43] to learn the weights of the edges of the graph.
After defining the model and describing some implementation details, we compare it to other state state of the art algorithms in both warm and cold start scenarios in three different dataset. Finally we analyze how the model is affected by users with short profiles.

# Chapter 2

# State of the art

In this chapter, we briefly describe why recommender systems are used and their basic structures. Then we'll focus on the state of the art algorithms used to provide recommendations. Finally, we introduce our new model.

## 2.1 Goals of recommender systems

Recommender systems are a subset of information filtering systems that try to predict users' preferences [6]. They are mostly utilized by merchants to increase product consumption by helping users navigate their large set of items in order to find new interesting products [42]. As a result, recommendations quality has a direct impact on user experience, which can be improved by making recommendations meet several operational goals. The most common ones are [2]:

- **Relevance**: the most important goal for a recommender system is to recommend items that are relevant to the target user. Users are more likely to consume items that they find interesting.
- **Novelty**: recommendations are useful only if the target user hasn't seen the recommended items yet.

- **Serendipity**: a recommender system should produce surprising result, recommendations that are unexpected. This helps the users to discover new types of items that they wouldn't interact with otherwise.
- **Diversity**: most recommendation problems consist of recommending a given number of items to a user. The latter should not be very similar, otherwise the user may not be interested in any of them.

## 2.2 Basic concepts of recommender systems

In the past few years, we observed a rapid growth of web based services, most of which make an extremely huge set of items available to the users. Recommender systems are used to help users navigate the products such that they can easily find items relevant to them [44], often, even without having to perform a search. The quality of recommendations can directly affect their user experience.
In order to produce good recommendations, recommender systems need a significant amount of data [2]. In fact, the rapid growth of recommender systems in the recent past is fueled by the ease with which feedbacks from users can be created and collected.

### 2.2.1 Data of a recommender systems

As hinted in the *Introduction*, the information used by recommender systems are mainly of two different types: collaborative and content. In this this section, we will explain how they are created and used by a recommender system.

#### Collaborative feedback

Collaborative information represents the interaction between users and items. For a recommender systems, it's the most important type of input

[6]; it allows to learn the users tastes and provide personalized recommendations. Most services allow their users to provide feedback for items they have interacted with. The feedback can be of four different types[44, 50]: a numerical rating within a range, an ordinal rating, a binary rating or a unary one. For a more compact representation, ordinal ratings and numerical ratings could be considered as a single type. Amazon, for example, allows users to rate products they have bought within a range from 1 to 5 stars. YouTube gives to its users the possibility to provide binary ratings through the *like* and *dislike* functions. Finally, Facebook allows users to like posts, thus providing unary ratings. These feedbacks are referred to as *explicit feedback*.
Collaborative information can be also collected implicitly, creating the so-called *implicit feedback*. While explicit ratings can come in different types, implicit ones can only be unary [2, 44]. They are created from the users behavior, without requiring any further action from them. For example, a service like Spotify can log for every user the songs they listened to, and use the information for future recommendation.
Explicit feedback are more general since they can always be transformed in unary ratings, if the used algorithm is designed for implicit feedback, like the model introduced in [28, 30, 43] . Another advantage of explicit ratings is that they allow users to provide negative feedback; a rating of 2 out of 5 probably means that the user didn't enjoy that particular item. Despite all these disadvantage, implicit feedbacks have similar performance to explicit ones [29]. This, combined with their easiness to create, makes them very common in recommender systems. Furthermore, their lack of expressiveness can be compensated with the quantity of data. Most users don't take the time to explicitly rate items they interacted with. They are more likely to express their preferences if they feel strongly for or against an item [3]. For example in the google Play Store it's very common to come across applications with more than 100 million downloads, but with way less than 10 million feedbacks.

Independently from the type of ratings, users' feedbacks are fed to the recommender system through the *User Rating Matrix, R.* The latter is a $m \times n$ matrix with $m$ equal to the number of users, $n$ equal to the number of items, and $Rui$ is the rating by user $u$ of item $i$.

### Content information

Content information describes the items by specifying their properties. The latter are called *features* and can be anything that characterize an item [44]. For example, for a movie, among all features we can have the director, the actors, the genres, etc.
This information is already available when the item is created, but some preprocessing is required to extract the descriptive features of the items. The most common approach is to extract keywords from the underlying data since unstructured text descriptions are often widely available in a variety of domains, and they remain the most natural representations for describing items [2]. A movie, for example, can be characterized by its title and synopsis.
Another type of content information is the one generated by users. Some services allow users to provide very short, usually just a single word, textual description, referred to as *tags.* Using tags, users can improve the quality of content data by providing descriptions that are more accurate and specific than the original features [16, 5]. While tags can considerably improve recommendations[2, 16], they have two main issues [24]: redundancy and ambiguity. The former is observed when synonyms are used to describe the same item, while the latter corresponds to the use of homonyms.
Also in this case, a matrix is used to represent the data: the *Item Content Matrix, C.* Usually it's a binary matrix, with the items as rows and the features as columns. If item $i$ has feature *f*, $C_{if}$ will be 1; otherwise it will be 0.

While collaborative and content information are the most used, recommenders can also rely on knowledge and demographic data. The former is made of explicitly specified user requirements [2], and is used mostly in scenarios where there is not enough feedbacks available [2, 10], while the latter consists of users' description [2].

### 2.2.2 The recommendation problem

The final goal of a recommender system is to predict the users ratings or preferences. While this can be done with several approaches, the problem is typically formulated in two ways[2]:

1. **The prediction problem**: With this approach, the goal of the system is to predict for every user the rating they would give to the items they haven't rated yet. As already mentioned, the ratings provided by users are saved in the URM, which is normally very sparse. This method consists of predicting the missing entries of the URM. In some cases, it is referred to also as the *matrix completion problem*: the ratings present in the matrix are used to predict, to fill, the empty part.

2. **The ranking problem**: It is the most common use case of recommender systems. To make recommendations to a user, it is not necessary to predict the rating they would give to every item. Usually, a service provider's objective is to recommend, for every user, a personalized list of items. This is accomplished by defining, for every user, an ordering of the items and recommending the first $k$ ones they haven't interacted with yet. That's why this approach is sometimes referred to as the *top-k recommendation problem*

The prediction approach is more general since the predicted ratings can be used to formulate personalized rankings, but, as we will see, there are methods designed to solve the ranking problem directly [43].

## 2.3 Collaborative filtering

Collaborative filtering is one of the most used approaches to design recommender systems [6]. The basic idea is that the observed ratings are often highly correlated across various users and items [2]. The recommending algorithm can exploit these correlations to infer the missing ratings. For example, let A and B be two similar users, with same tastes. They will probably give similar ratings to the same items. Thus, the system can use the ratings of A to predict the ratings of items seen by A but not by B, and vice versa.
There are two types methods used to perform collaborative filtering: memory based and model based [2, 44].

### 2.3.1 Memory based CF

Memory based collaborative filtering algorithms, also referred to as *neighborhood based*, use users' ratings to compute similarity between users or items to provide recommendations. The ratings of users, or items, are predicted based on their neighborhood [2, 44, 6]. Memory based algorithms can solve the recommendation problem with two approaches: user based collaborative filtering and item based collaborative filtering.

#### User based CF

In user based CF, the basic idea is that similar users rate the same item in similar ways [2]. To predict the rating by user $u$ of item $i$, a weighted average is performed over the ratings by users most similar to $u$ on the same item. The similarities between $u$ and the other users are used as weights.

$$R'_{ui} = \frac{\sum_{v \in U} R_{vi} \cdot sim(u,v)}{\sum_{v \in U} sim(u,v)} \tag{2.1}$$

where $sim(u,v)$ is the similarity between user $u$ and user $v$.

### Item based CF

In item based CF, the basic idea is that a user give similar ratings to similar items [2]. While with user based the prediction of a rating is computed from the ratings given by other users on the same item, with item based, instead, the ratings of the target user on similar items are used.

$$R'_{ui} = \frac{\sum_{j \in I_u} R_{uj} \cdot sim(i,j)}{\sum_{j \in I_u} sim(i,j)} \tag{2.2}$$

### Comparing user based CF to item based CF

Item based CF tends to produce more accurate predictions, since it is based on the target user's own ratings [2]. The user based approach suffers from the fact that users ratings can have different biases. For example, there are users that give high ratings in general, and other users with usually low ratings. Furthermore, the produced recommendations can be easily explained [2, 44]. Users are more likely to consider a recommendation if they know the reason behind it. For example, on Netflix someone can find the following recommendation: *Because you liked Breaking bad, we recommend you Better call Saul.* This type of explanations can be easily provided with an item based approach.
The problem with item based CF, though, is that it produces recommendation of items very similar to the ones that the target user has already seen [2, 6], leading to a low serendipity. In fact, it is not able to recommend items that don't share any feature with the target user's items. User based CF doesn't have this problem. An item completely different from the target user's items can be recommended if it is liked by very similar users.

## 2.3.2 Model based CF

The basic idea behind model based collaborative filtering is that the observed ratings can be represented, or approximated, by a mathematical model [2]. The approach can be summarized into two phases: a training phase and a predictive one. The training phase, also referred to as the model building phase, consists of building a model that summarizes the observed ratings. This is done by fitting the generic model to the given dataset using some machine learning or data mining techniques[2]. Depending on the chosen model and on the dataset size, this phase can be quite computationally expensive. The predictive phase consists of using the model generated during the training phase to predict user ratings in order to make recommendations. Unlike neighborhood based methods, once a model is obtained, it can be used to make recommendations without needing the whole dataset every time. As a result, the system can improve in speed and scalability [2, 6, 1]

### The overfitting problem

Model based algorithms rely on a mathematical model to provide recommendations. These models use the training data, which in case of collaborative filtering algorithms is the URM, or a portion of it, to fit a generic model with the actual data. The final result is generally an approximation of the URM[2]. However, in some cases the model fits excessively the training data. In these cases, the result is an extremely close approximation of the URM used in the training phase and will produce poor recommendations since it would put a 0 in missing cells of the URM [2]. The problem is referred to as overfitting [26]. Many models in recommender systems literature try to attenuate the overfitting effects by applying a regularization during the training[20, 39, 41, 43].

### Matrix factorization

Latent factor models try to explain the ratings by characterizing both items and users on factors inferred from the ratings patterns. Some of the most successful latent factor implementations are based on matrix factorization [32].

Matrix factorization maps both users and items in the same latent factor space of dimensionality $k$, which represents the $k$ inferred factors [2, 32]. In its basic model, the $m \times n$ URM is approximately factorized into an $m \times k$ matrix $U$ and an $n \times k$ matrix $V$ as follows:

$$R' = U \odot V^T \tag{2.3}$$

Every user (item) is represented by a vector of size $k$ referred to as user factor (item factor). These vectors constitute the rows of $U$ $(V)$ and represent the affinity between the corresponding user (item) and the $k$ concepts in the URM [2]. Rating between user $u$ and item $i$ can be approximated by the dot product of the corresponding user factor and item factor:

$$R'_{ui} = U_u \odot V_i^T = \sum_{j=1}^{k} U_{uj} \cdot V_{ij}^T \tag{2.4}$$

One of the main advantages of matrix factorization is its scalability [46]. The number of latent features is usually very limited and doesn't grow with the number of users or item. Therefor, even with huge sets of users or items, the model size will still be very compact. This also improves the computational complexity of the prediction phase. To provide recommendation, instead of using a huge dataset, the much smaller model is used [2, 6]. For instance, with a user-based neighborhood algorithm, the time complexity is $\mathcal{O}(|U|)$ since we need to go over all the users. With a matrix factorization model, on the other hand, we have a constant time complexity given that $k$ doesn't depend on the number of users.

### SVD

Singular Values Decomposition is one of first [46] and most popular techniques used to perform matrix factorization [32]. With SVD, the columns of $U$ and $V$ are constrained to be mutually orthogonal [2]. This allows to discover unique concepts that explain the observed ratings, thus leading to a more compact model. The main problem of SVD is that its result is undefined when input matrix is not complete [32]. As already stated, the URM is usually very sparse. Initially, to solve this problem, other algorithms were used to fill the missing ratings before applying SVD [32]. This technique is computationally expensive, requires a lot of memory and the final result depends heavily on the quality of the algorithm used to fill the URM. To overcome these drawbacks, other solutions have been developed [41]. One of the most famous one is *Funk SVD* [20], which emerged during the Netflix competition. Instead of filling the URM, Funk used only the observed data to fit the model, while adding regularization to avoid overfitting. This approach initializes randomly U and V and uses gradient descent to minimize the prediction error over the observed data:

$$e_{ui} = R_{ui} - R'_{ui} \tag{2.5}$$

$$U_{uk} = U_{uk} + \eta(e_{ui} \cdot V_{ik} - \lambda \cdot U_{uk}) \tag{2.6}$$

$$V_{ik} = Vik + \eta(e_{ui} \cdot U_{uk} - \lambda \cdot V_{ik}) \tag{2.7}$$

where $\eta$ is the learning rate and $\lambda$ is the regularization factor.

### Alternate least square

Alternate least square (ALS) can be used as alternative to stochastic gradient descent as optimization technique. While stochastic gradient descent is simple and very efficient, it suffer from being sensitive to initialization and the choice of the update step [2]. ALS provides a more stable solution which can be summarized as follows:
After initializing $U$ and $V$, theses two steps are repeated until convergence is reached [2, 32]:

1. Keep $U$ fixed and solve each row of $V$ by treating the problem as a least-squares regression problem, using only the observed ratings. The optimal value of the $i$th row of $V$ is found by minimizing

$$\sum_{u \in R_i} (R_{ui} - \sum_{l=1}^{k} U_{ul} \cdot V_{il})^2 \tag{2.8}$$

with $U_{ul}$ treated as constant, for each $u$ and $l$, and $V_{il}$ as optimization variable, for each $l$.

2. Keep $V$ fixed and solve each row of $U$ by treating the problem as a least-squares regression problem, using only the observed ratings. The optimal value of the $u$th row of $U$ is found by minimizing

$$\sum_{i \in R_i} (R_{ui} - \sum_{l=1}^{|k|} U_{ul} \cdot V_{il})^2 \tag{2.9}$$

with $V_{il}$ treated as constant, for each $i$ and $l$, and $U_{ul}$ as optimization variable, for each $l$.

In both steps, since the least-squares regression problem for each row is independent, the process can be parallelized [2, 32].

**SLIM**

Sparse linear models (SLIM) are introduced in [37] to perform top-N recommendations. The rating of an unseen item $i$ by user $u$ is computed as a sparse aggregation of items that have been already rated by $u$:

$$R'_{ui} = \sum_{j \in I} R_{uj} \cdot W_{ij} \tag{2.10}$$

Thus, the complete model can be obtained as:

$$R' = R \odot W \tag{2.11}$$

where $W$ is an $|I| \times |I|$ matrix of aggregation coefficients. It is learnt by minimizing the following expression over the observed ratings:

$$\frac{1}{2}||R - R \odot W||_F^2 + \frac{\beta}{2}||W||_F^2 + \lambda||W||_1 \tag{2.12}$$

subject to the constraints $W >= 0$, to avoid producing negative ratings, and $diag(W) = 0$, to avoid a solution in which $W$ is the identity matrix. A hybrid extension, SSLIM [38], has been proposed by Ning et al. which involves both the use of collaborative and content information.

**Bayesian Personalized Ranking**

As stated in section 2.2.2, the recommendation problem can be formulated either as a prediction problem or as a ranking problem [2]. Most models, like the ones presented so far, use the prediction approach [2], even though the final goal, in most cases, is to produce a personalized, ranked, list of items for each user.
Rendle et al. [43] introduced Bayesian Personalized Ranking (BPR) to solve the ranking problem directly, to learn to rank. It is a generic learning algorithm based on stochastic gradient descent[8]. From the observed ratings, they try to reconstruct the complete preference order of the users by considering the missing ratings as negative. More precisely, they assume that a user $u$ prefers a seen item $i$ over all unobserved items $j$ [43]. With $x_{u,i}$ the score of user $u$ of item $i$, the score difference $x_{u,ij}$ is defined as:

$$x_{u,ij} = x_{u,i} - x_{u,j} \tag{2.13}$$

From this formula, the probability that user $u$ really prefers item $i$ over item $j$ can be defined by using the logistic sigmoid function [14]:

$$p(i >_u j|\Theta) = \frac{1}{1 + e^{-x_{u,ij}}} \tag{2.14}$$

where $\Theta$ is the vector of parameters.
The basic idea of BPR is to use stochastic gradient descent to maximize this probability. The parameter vector $\Theta$ can be updated with the following formula:

$$\Theta = \Theta + \alpha \left( \frac{1}{1 + e^{-x_{u,ij}}} \cdot \frac{\partial x_{u,ij}}{\partial \Theta} + \lambda \Theta \right) \tag{2.15}$$

where $\alpha$ is the learning rate and $\lambda$ is the regularization factor.

## 2.4 Content based

Content based recommender systems rely on the description of items to make recommendations. The basic idea is to recommend to users items

that are similar to the ones they already like [2]. For example, if a user liked both *The godfather* and *Goodfellas* movies, we can assume that they will also like *The departed* since they are very similar. Given that items' descriptions are always available, this type of recommendation can be performed also with items recently added into the system, without any ratings yet.
One of the most used content based implementations rely on nearest neighbour algorithms [2, 6].
To predict the rating of user $u$ of item $i$, a similarity measure is used to find the most similar items to $i$. Finally, the rating that $u$ gave to those items are used to compute the rating of user $u$ on item $i$.

$$R'_{ui} = \frac{\sum_{j \in I_u} R_{uj} \cdot sim(i,j)}{\sum_{j \in I_u} sim(i,j)} \tag{2.16}$$

## 2.5 Similarities

Many collaborative filtering and content based recommender systems require a similarity or distance function to operate [44]. Based on the type of recommender system, the similarity, or distance, can be computed either between users or between items [2, 44].
The following measures are the most common ones.

### 2.5.1 Cosine similarity

Cosine similarity measures, as the name suggests, the cosine of the angle between two vectors [44]. Given $x$ and $y$, two n-dimensional vectors, their cosine is defined as:

$$cos(x,y) = \frac{x \odot y}{||x|| \cdot ||y||} \tag{2.17}$$

where $\odot$ represents the dot product between the two vectors and $||x||$ is the norm of the vector $x$. Note that, considering the possibility of having negative values in the vectors, the cosine similarity ranges from -1, not similar at all, to $+1$, identical.

### 2.5.2 Jaccard similarity

Jaccard similarity measures the similarity between two sets by comparing the number of elements present in both sets to the total number of elements in either sets [2]. Given two sets $X$ an $Y$, the jaccard similarity is defined as:

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{2.18}$$

Since the cardinality of a set is always non-negative and $|A \cup B| \geq |A \cap B|$, follows that $0 \leq J(X,Y) \leq 1$.

### 2.5.3 Pearson correlation

The Pearson correlation coefficient measures the linear correlation between two objects [44]. Given $X$ and $Y$ two random variables, the Pearson correlation $\rho$ is defined as:

$$\rho_{\mathrm{X,Y}} = \frac{cov(X,Y)}{\sigma_{\mathrm{X}}\sigma_{\mathrm{Y}}} \tag{2.19}$$

where $cov$ is the covariance and $\sigma_{\mathrm{X}}$ and $\sigma_{\mathrm{X}}$ are the standard deviations of $X$ and $Y$ respectively. Applied to two vectors of dimension $n$, the formula becomes:

$$sim(x,y) = \frac{\sum_{i=1}^{n}(x_{\mathrm{i}} - \bar{x})(y_{\mathrm{i}} - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_{\mathrm{i}} - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_{\mathrm{i}} - \bar{y})^2}} \tag{2.20}$$

where n is the size of the vectors $\bar{x}$, and $\bar{y}$ are the sample means of x and y respectively. Note that the Pearson coefficient ranges from -1, total negative linear correlation, to 1, total positive linear correlation, while 0 means no linear correlation.

## 2.6 Graph-based methods

Graph based approaches use a graph to represent the data. The set of nodes consists of the union of the user set, item set and feature set [44].

The relations between these nodes are represented by the edges. Since in the data used by recommender systems we usually have two kind of relations, user-item relations and item-feature relations, also the graph has two types of edges: edges from user nodes to item nodes and edges from item nodes to feature nodes.
Edges can have associated weights to represent the importance of the relation between two nodes [2]. In case of undirected graphs, there can be only a unique edge between two nodes. With a directed graph, instead, between two nodes, we can have two different edges. This allows to give different weights, importance, to the edges from node $A$ to node $B$ and from $B$ to $A$.
Nodes that are not directly connected influence each other by propagating information along the edges of the graph. The weight of an edge controls how much information can pass through; the greater the weight of an edge, the more information is allowed to pass through it [2].
In this section we will present some concepts used in graph based models and two of the most popular models.

### Adjacency matrix

An adjacency matrix is an $n \times n$ matrix, with $n$ equal to the number of nodes. It has a row and a column for every node in the graph and indicates whether two nodes are directly connected to each other with an edge. Usually the adjacency matrix, $A$, is binary and is defined as:

$$A_{ij} = \begin{cases} 1 & \text{if node } i \text{ and node } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

With a weighted graph, a variant of the adjacency matrix can be use where, instead of a binary representation we can use the edges weights to represent adjacency. Said matrix can be defined as:

$$A'_{ij} = \begin{cases} w_{ij} & \text{if node } i \text{ and node } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

where $w_{ij}$ is the weight of the edge between node $i$ and node $j$.

**Random walk**

Many graph based methods uses the concept of random walk [13, 11, 33, 19]. It is a process that describes a walker that passes through the graph choosing, at every step, the next node randomly [11]. It corresponds to a first-order Markov process defined by a set of $n$ states, corresponding to the $n$ nodes of the graph, and an $n \times n$ transition matrix $P$. Thus, it has to satisfy the Markov property, meaning that, at any step, the probability of reaching a new state depends only on the current one, it cannot be affected by the history of the walker. The transition matrix, $P$, can be obtained by normalizing row by row the adjacency matrix $A'$:

$$P_{ij} = \frac{A'_{ij}}{\sum_{k \in V} A'_{ik}} \tag{2.23}$$

where $V$ is the set of nodes and $i, j \in V$

### 2.6.1 $\mathbf{P^3_\alpha}$

$P^3_\alpha$ is a generalization of the $P^3$ algorithm. Due to their collaborative nature, the feature nodes will not be considered in the graph. As a result, the latter is a bipartite graph containing only the user nodes and the item nodes.

$P^3$ ranks the items for a user $u$ according to the distribution of the third vertex of the random walk starting at vertex $u$ [13]. Equivalently, the name stands for the third power of the transition matrix P. Other powers can be used instead of 3, but it has been shown that higher values produce lower performances [13], and random walks of length less than 3 are meaningless given that walks of length 1 lead to the target user's own item, while walks of length 2 lead to user nodes.

$P^3_\alpha$ generalises $P^3$ by introducing a new parameter, $\alpha$. From the transition matrix $P$, we can compute $P_\alpha$ by raising the positive values to the power of

$\alpha$.

$$P_{\alpha,ij} = \left(\frac{A'_{ij}}{\sum_{k\in V} A'_{ik}}\right)^{\alpha} \tag{2.24}$$

where $\alpha$ is a real number.
$P^3_\alpha$ uses the third power of $P_\alpha$ to rank items for recommendation [13].
In a collaborative algorithm, due to the absence of edges between nodes of the same type, the cells of the transition matrix $P$ can be different from 0 only in two blocks. The first one represents edges from users to items and can be computed by normalizing the URM row by row, while the second one represents the edges from items to users and can be computed by normalizing the transpose of the URM row by row. By applying the definition of $P_\alpha$ to these blocks, we get:

$$P_{ui,vj} = \left(\frac{R_{vj}}{\sum_{l\in I} Rvl}\right)^{\alpha} \tag{2.25}$$

and

$$P_{iu,jv} = \left(\frac{R_{vj}}{\sum_{s\in U} Rsj}\right)^{\alpha} \tag{2.26}$$

With $\text{P}^3{}_\alpha$, the probabilities to reach an item $i$ from a user $u$ can be computes as:

$$R'_{vj} = P_{ui,v} \odot P_{iu} \odot P_{ui,j} \tag{2.27}$$

$$R' = P_{ui} \odot P_{iu} \odot P_{ui} \tag{2.28}$$

### 2.6.2 $\mathbf{RP^3_\beta}$

Paudel et al. [11] observed that, in recommendations based on the the transition matrix probability $P^3$, the popularity of the items has a significant influence on their ranking. This promotion of already popular items led to the concern that recommender systems may reinforce the blockbuster nature of media [18].
To promote the recommendation of items from the long tail, they proposed a new model, $RP^3_\beta$. The basic idea is to re-rank the results obtained from

$P^3$ based on items popularity [11]. Thus, the score of item $i$ from user $u$ is computed as:

$$R'_{ui} = \frac{P^3_{ui}}{D_i^{\beta}} \tag{2.29}$$

where $D_i$ is the in-degree, the number of incoming edges, of item $i$.
For two items $i$ and $j$ with $P^3_{ui} = P^3_{uj}$ and $D_i < D_j$, meaning that $j$ is more popular than $i$, this model will rank $i$ higher than $j$. The parameter $\beta$ can be used to control the magnitude of the re-ranking process: a value of 0 means that there is no re-ranking, while larger values mean larger penalty for popular items.

## 2.7 Hybrid recommender systems

As we've previously mentioned, the most popular recommender algorithms can be classified in three different classes [2] based on the type of data they use to provide recommendations. They have different strengths and weaknesses.
Usually collaborative filtering algorithms provide better performance than content based ones [2, 6, 44]. Thanks to the collaborative power of ratings, they are also able to provide a more diverse recommendation list [2, 6].
One of the main problem of content based algorithms is that they recommend items that are very similar to the ones already rated by the user [6]. In fact, if an item doesn't have some features in common with the target user's items, it is unlikely that it will ever be recommended to that user [2]. This leads to a low diversity and serendipity, enclosing the users in filter bubbles [40] by exposing them to only a slightly narrowing set of items over time [36].
Collaborative filtering, on the other hand, is not able to provide recommendations in a cold start scenario [2, 44]. Content based algorithms can recommend newly added items as long as there are user that have rated similar items. This makes them extremely useful in systems with a

dynamic set of items [2].
Collaborative filtering and content based systems complement each other. Collaborative algorithms provides very good recommendation if enough ratings are available while content based ones are able to recommend also cold-start items. This is the reason they are sometimes combined into a hybrid system [48]. Thus, using the advantages of a class to compensate the disadvantages of another [44].
In this section we will go over the most used techniques to create hybrid recommender systems according to Burke [9].

### 2.7.1 Weighted hybrids

With a weighted hybrid, the score of an item is computed by combining the scores of different recommender systems. A simple linear combination can be used. This technique is very straightforward to implement and makes possible the use of recommender systems out of the box.
The main drawback of this technique is that the different type of algorithms don't have the same value in every situation [9]. For example, in a weighted hybrid between a CF and a CBF, the CF would have assigned a larger weight since it has better performance in general. However, in a cold start scenario, the CBF should have a higher value, but this is not taken into account.

#### CF-CBF classic hybrid

The *CF-CBF classic hybrid* uses a weighted approach to combine collaborative filtering with content based filtering. The basic idea consists of using the users' ratings as items' descriptions by concatenating the transpose of the URM with the ICM, which is multiplied by a weight.

$$F = R^T || C \cdot w_c \tag{2.30}$$

where $C$ is the ICM. The weight $w_c$ can be used to give more, or less, value to the content part.

Then, a neighborhood algorithm is used on the item similarities computed from $F$ to provide recommendations.

### 2.7.2 Switching hybrids

A switching hybrid uses a defined criterion to switch between different recommender systems according to the context. In this case the implementation can be a bit more challenging given that the switching logic must be implemented. However, it is still possible to use single components out of the box and the switching mechanism allows the systems to be more sensitive to the strengths and weaknesses of the single components.
This technique can be used to tackle the cold start scenario for new users [9]. Recommender systems can use a knowledge based system to provide recommendations to new users, and subsequently switch to a collaborative algorithm once the user has several ratings, like the algorithm proposed in [10].

### 2.7.3 Mixed hybrids

Mixed hybrids are typically used in situations in witch it is necessary to provide a large number of recommendations. Items recommended by different systems are simultaneously recommended to the user. A CF and CBF mixed hybrid would be able to find items very similar to the ones of the target user, via the CBF, while also introducing a degree of novelty, due to the CF, without having to compromise between the two.

### 2.7.4 Cascading hybrids

With a cascading hybrid, a first recommender is used to provide a ranking of candidate items. A second recommender is then used to refine the ranking of the candidates. This operation can be repeated with different

recommenders. The training of a component is biased by the output of the previous one.
An example of cascading hybrid has been proposed in [25] in which collaborative and demographic correlations are applied over the candidate neighbour items found with the content data.

### 2.7.5 Feature augmentation hybrids

In feature augmentation hybrids, one technique is employed to produce a rating or a classification on an item and that information is then incorporated into the processing of the next recommendation technique [9].
An example of augmented hybrid is proposed by Cremonesi et al. in [15] for new item recommendation. A content based algorithm is used to predict a portion of the missing ratings. The latter are then combined with the observed rating and fed to a collaborative algorithm.

## 2.8 Evaluation of recommender systems

Recommender systems can be evaluated using either online or offline methods [2].
Online evaluation measures the reaction of the user with regard to the presented recommendations. One of the most used online evaluation technique is *A/B testing*[23]. It consists of measuring the conversion rate of users clicking, or more generally consuming, the recommended items in two different configuration, $A$ and $B$. This type of evaluation allows to directly measure the effectiveness of the system [2], but, since it requires users interaction, it is not feasible to use it in the research and development phase. Therefore, offline evaluation is usually used.
Even though there is an abundant number of metrics, evaluating recommender systems is still a difficult task [2, 27]. For example different

algorithms may perform better, or worse, depending on the datasets. Another challenge is the selection of the appropriate metrics. Most recommender algorithms are developed for a given goal, and based on the latter, some metrics become more relevant than others.
However, the principal goal of a recommender system is to find good items, items that are relevant to the target user. In order to do so, the system should be able to accurately predict the ratings of the users. In this section, we'll go over the most popular metrics used to measure the accuracy of a recommender system.

### 2.8.1 Accuracy metrics

Recommender systems are mainly used to find the preference order of users over the item set. Accuracy metrics empirically measure how close the predicted preference order for a user differs from the user's true preference order [27]. Based on how the difference is interpreted, the metrics can be classified into three distinct classes: predictive accuracy metrics, classification accuracy metrics and rank accuracy metrics.

#### Predictive metrics

Predictive accuracy metrics measures how close the predicted ratings are to the users' actual ratings [2, 27]. They are very important in systems in which the predicted ratings are shown to the users, but, as previously stated, this type of scenario is not very common in recommender systems. *Mean absolute error* (MAE) is one of the most used predictive accuracy metric [27]. It measures how close the predicted ratings are to the real one by computing the average absolute error [2, 27].

$$MAE = \frac{1}{|R|} \sum_{R_{ui} \in R} |R_{ui} - R'_{ui}| \tag{2.31}$$

### Classification metrics

Recommender system are mainly used to provide new, relevant, items to users. This can be considered as a binary classification problem in which, for every user, the system classifies the items either as relevant or as not relevant [44]. The possible outcomes of the classification can be represented in a $2 \times 2$ table called confusion matrix.

| | **Recommended** | **Not Recommended** |
|---|---|---|
| **Relevant** | true positive (tp) | false negative (fn) |
| **Not relevant** | false positive (fp) | true negative (tn) |

Table 2.1: Confusion matrix of a recommender system

where:

- *true positive* is an item correctly classified as positive
- *false positive* is an item incorrectly classified as positive
- *true negative* is an item correctly classified as negative
- *false negative* is an item incorrectly classified as negative

Classification accuracy metrics measures the frequency with which items are correctly classified [27]. Precision and recall are one of the most used metrics to evaluate information retrieval systems [27].
*Precision* is defined as the ratio of relevant items recommended to number of recommended items [27].

$$Precision = \frac{tp}{tp + fp} \tag{2.32}$$

It represents the probability that a recommended item is relevant [2, 27].
*Recall* is defined as the ratio of relevant items recommended to the number of relevant items[27].

$$Recall = \frac{tp}{tp + fn} \tag{2.33}$$

It represents the probability that a relevant item is recommended [2, 27].

### Rank metrics

Rank accuracy metrics measure the ability of a recommender system to produce a preference order of items that matches how the user would have ordered the same items. They are used to evaluate algorithms that will be used to present ranked recommendation lists to the user, in domains where the user's preferences in recommendations are non-binary [27].
*Mean Average Precision* (MAP) is one of the most used rank metrics [27, 47]. It computes the overall precision of the system at different lengths of recommendation lists [47]. MAP is computed as the arithmetic mean of the average precision over the entire set of users in the test set. Average precision for the top K recommendations, AP@k, is defined as follows:

$$AP@k = \frac{1}{N}\sum_{i=1}^{k} Precision@i \cdot Rel(i) \tag{2.34}$$

where $N$ is the number of relevant items and $Rel(i)$ is 1 if the $i^{th}$ recommended item is relevant, 0 otherwise. From this definition, we can compute MAP as:

$$MAP@k = \frac{1}{k}\sum_{n=1}^{k} AP@n \tag{2.35}$$

## 2.8.2 Beyond accuracy

Recommender systems must provide not just accuracy, but also usefulness [22]. For instance, a recommender might achieve high accuracy by only computing predictions for popular items, due to the amount of feedbacks available for the latter. However, those items are the less likely to need recommendation [27]. Evaluation of recommender systems must also move beyond accuracy to include suitability of the recommendations to users.

### Coverage

The coverage of a recommender system is a measure of the domain of items in the system over which the system can generate predictions or make

recommendations [27].
An example of coverage measure is *prediction coverage* [2, 27]. It represents the number of items for which predictions can be formed as a percentage of the total number of items. It is possible for a recommender to achieve a high coverage by predicting inaccurate ratings for a portion of the items. Therefore, coverage measures should always be accompanied with accuracy metrics [2, 27]

### Novelty

Because of the amount of ratings available for popular items, recommenders are able to score a high accuracy by predicting them. These obvious recommendations, though, are not very useful to the final user since the selected items could be easily discovered without the help of the system.
Novelty metrics allow to measure the "non-obviousness" of the system [27], the ability of a recommender to introduce users to items that they have not previously experienced in real life.
A user's average novelty over the items of their recommended list $\mathrm{L_u}$ can then be defined as [47]:

$$novelty = \frac{1}{|U|} \sum_{u \in U} \sum_{i \in L_u} \frac{-log_2(pop_i)}{|L_u|} \tag{2.36}$$

where $pop_i$ is the popularity of item $i$, measured as percentage of users who rated $i$.

## 2.9 Introducing our work

The graph based recommenders described in the previous sections, $P^3_\alpha$ and $RP^3_\beta$, don't use the content information. Due to their pure collaborative nature, they are not able to produce recommendation in a cold start scenario. Moreover, every edge of the graph has the same weight. Thus, the random walk they rely on uses a uniform distribution to select the next node in the walk. While this technique is quite simple to implement, it doesn't fully exploit the information contained in the data. For instance, with a movie graph based, hybrid, recommender, if the transition from the item nodes to the feature nodes has a uniform distribution, the probability to reach a *genre* node, or a *director* node, is equal to the probability to reach a *production year* node. Every node, and every edge, has the same importance, even though, in practice, they have different levels of relevance.

In this thesis, we overcome these two drawbacks by proposing a new hybrid graph based model. It uses both collaborative and content data to provide recommendations with three steps random walks. Thereby, even if an item does not have any rating yet, it can still be reached from a user node by passing through a feature node.

In order to fully exploit the information within these two classes of data, we employ a machine learning optimization algorithm, BPR, to learn the weights of a portion of the edges of the graph. This allows to abandon the uniform distribution, hence selecting the next node based on its relevance, and to find an adequate balance between the collaborative portion and the content one.

As a result, the model is able to provide recommendations in cold start scenario and to take into account the different level of importance between the nodes of the graph.

# Chapter 3

# The model

In this section we will focus on defining our model and how BPR is used to learn the parameters. Finally, we will provide the implementational details.

## 3.1 An unweighted graph approach

In order to easily explain our model, we initially use an unweighted graph. Meaning that it relies on random walks with uniform distributions.
As already mentioned in the previous section, the proposed model is a graph based collaborative-content hybrid. Therefore, the graph has three different types of nodes:

- user nodes
- item nodes
- feature nodes

and four types of edges:

- from a user node to an item node
- from an item node to a user node

- from an item node to a feature node
- from a feature node to an item node

Since in this section the considered graph is unweighted, bidirectional edges can be used. An example of graph that combine the two edges between the same two nodes can be observed in figure 3.1.

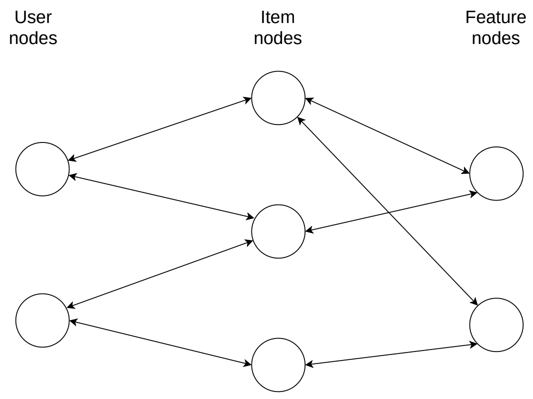
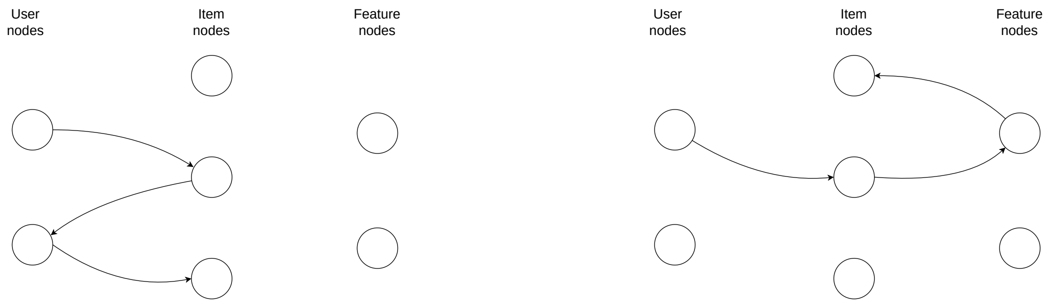


Figure 3.1: An example of graph

Recommendations are based on three steps walks, starting from a user node and ending into an item node. Consequently, there are two possible types of paths that can lead to a recommendation: the collaborative path and the content path.



(a) Collaborative path

(b) Content path

Figure 3.2: Possible paths in a three steps walk

In both cases, the first step consists in passing from a user node to an item node. This operation is a one step random walk characterized by the transition probability matrix $P_1$, which can be computed by normalizing row by row the URM. $P_{1,ui}$ represents the probability of reaching the item node $i$ from the user node $u$, and can be computed as:

$$P_{1,ui} = \frac{R_{ui}}{\sum_{j \in I} R_{uj}} \tag{3.1}$$

where $I$ is the set of items. From this point, the two types of paths diverge.

- **collaborative path**: the second step of the collaborative path consists in passing from an item node back to a user node. Also in this case, the operation can be represented by a one step random walk. Its transition probability matrix, $P_{2'}$, can be obtained by normalizing row by row the transpose of the URM. The probability of reaching the user node $u$ from the item node $i$, $P_{2',iu}$, can be computed as:
$$P_{2',iu} = \frac{R_{ui}}{\sum_{v \in U} R_{vi}} \tag{3.2}$$
where $U$ is the set of users.
The third and last step is equal to the first one, and its transition probability matrix, $P_{3'}$, can be computed in the same way as $P_1$.
Finally, we can write the probability of user $u$ to reach item $i$ by following the collaborative path, $x'_{ui}$, as:
$$x'_{ui} = P_{1,u} \odot P_{2'} \odot P_{3',i} \tag{3.3}$$

- **content path**: with the content path, on the other hand, the second step is the passage from an item node to a feature node. As with the previous cases, it can be formalized with a one step random walk represented by the transition probability matrix $P_{2''}$, which can be obtained by normalizing the ICM row by row. $P_{2'',if}$ represents the probability of reaching the feature node $f$ form the item node $i$, and can be computed as:

$$P_{2'',if} = \frac{C_{if}}{\sum_{h \in F} C_{ih}} \tag{3.4}$$

where $F$ is the set of features.
Similarly, the last step consists in passing from a feature node to an item node. The transition probability matrix representing this step, $P_{3''}$, can be obtained by normalizing row by row the transpose of the ICM. The probability of reaching an item node $i$ from a feature node $f$, $P_{3'',fi}$, can be computed as:

$$P_{3'',fi} = \frac{C_{if}}{\sum_{j \in I} C_{jf}} \tag{3.5}$$

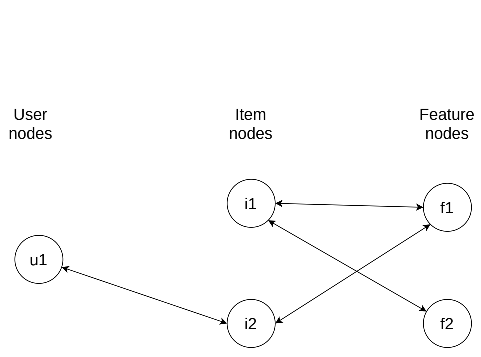
Therefore, the probability of reaching node $i$ from user $u$ via the content path, $x''_{ui}$, can be written as:

$$x''_{ui} = P_{1,u} \odot P_{2''} \odot P_{3'',i} \tag{3.6}$$

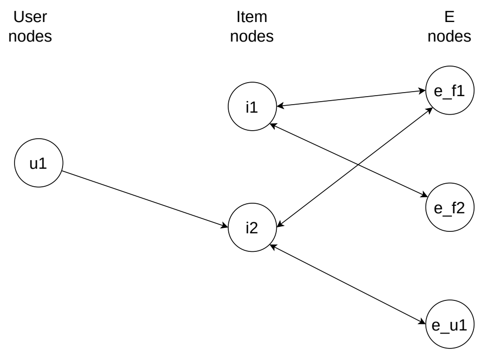
To merge the collaborative and the content paths, we unified the user nodes and the feature nodes into a single type of node, referred to as $E$ nodes, such that from an item node, there is only one possibility: to go to an $E$ node. The adjacency matrix between the item nodes and the $E$ nodes, referred to as UFM from now on and represented by the letter $T$, can be obtained by concatenating the ICM with the transpose of the URM.

$$T_{ie} = \begin{cases} R_{ei} & \text{if } e \in U \\ C_{ie} & \text{if } e \in F \end{cases} \tag{3.7}$$

As a result, there is only a single type of path that, from a user node, leads to an item node with a three steps walk. An example of the resulting graph can be observed in figure 3.3.



(a) original graph



(b) graph with $E$ nodes

Figure 3.3: Graph transformation

The first step, from a user node to an item node, remains unchanged, and is still characterized by the transition probability matrix $P_1$.
Now the second step consists in passing from an item node to an $E$ node. However, given that the $i^{th}$ row of the UFM contains both features and users of item $i$, the transition probability matrix of this step cannot be obtained by normalizing the UFM row by row because of the large cardinality difference that can be observed between users and features. For example, imagine to have a very popular item. Because of the large number of users who rated it, by normalizing row by row, the content part becomes negligible due to the large denominator. The opposite can also occur. For a movie recommender system, for example, we can have a large number of features because of the list of actors, directors and crew members.
This is the reason why we decided to compute the transition probability matrix $P_2$ by normalizing the collaborative part and the content one separately. Note that because of this, the sum of the transition probabilities from an item node toward $E$ nodes is equal to 2. So in order for the result be an actual transition probability matrix, it should be divided by 2. However, since it doesn't affect the model, we decided to omit it. Finally, we can write $P_{2,ie}$, the probability of reaching an $E$ node $e$

from an item node $i$, as:

$$P_{2,ie} = \frac{T_{ie}}{g_{ie}} \tag{3.8}$$

where $g_{ie}$ is defined as:

$$g_{ie} = \begin{cases} \sum_{v \in U} R_{vi} & \text{if } e \in U \\ \sum_{h \in F} C_{ih} & \text{if } e \in F \end{cases} \tag{3.9}$$

The third step, represented by the transition probability matrix $P_3$, can be obtained by normalizing row by row the transpose of the UFM. The probability of reaching the item node $i$ from the $E$ node $e$, $P_{3,ei}$, can be computed as:

$$P_{3,ei} = \frac{T_{ie}}{\sum_{j \in I} T_{je}} \tag{3.10}$$

Finally, the probability of reaching item node $i$ from user node $u$ can be written as:

$$x_{ui} = P_{1,u} \odot P_2 \odot P_{3,i} \tag{3.11}$$

## 3.2 Adding weights into the graph

In the model described in 3.1, the graph is unweighted, so the random walks rely on uniform distributions. As already mentioned, this type of approach is not able to fully exploit the available information since it does not take into account the different levels of relevance of the nodes and edges. To overcome this issue, we based our model on a weighted graph. Every $E$ node has an assigned weight, and it is proportional to the probability to reach said node from an item node. The reason we did not assign the weights directly to the single edges is because of their numerosity. Having a parameter for every edge from an item node to an $E$ node would be very inefficient.
Therefore, only the outgoing edges of item nodes are weighted, and the weight value depends on the destination node. In other words, all incoming

edges of a given $E$ node have the same weight, the weight of that particular $E$ node.
An example of the weighted graph can be observed in figure 3.4.

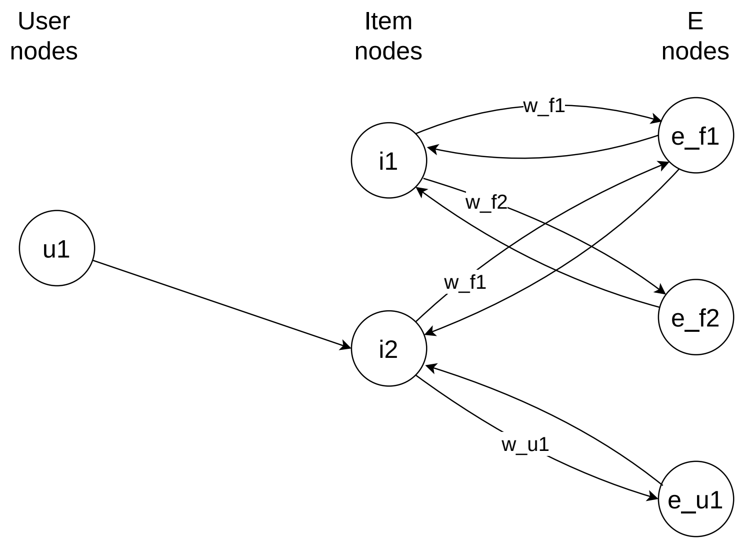


Figure 3.4: An example of weighted graph

The first and third steps of our three steps random walks are not affected by the new graph. They are still characterized by the already defined transition probability matrices $P_1$ and $P_3$, respectively.
For the second step, on the other hand, a uniform distribution can no longer be used given that the corresponding portion of the graph is now weighted. With $w_e$ being the weight assigned to the $E$ node $e$, we can re-define the probability of node $e$ to be reached from the item node $i$, $P_{2,ie}$, as:

$$P_{2,ie} = \frac{T_{ie} \cdot w_e}{g_{ie}} \tag{3.12}$$

where $g_{ie}$ is defined as:

$$g_{ie} = \begin{cases} \sum_{v \in U} R_{vi} \cdot w_v & \text{if } e \in U \\ \sum_{h \in F} C_{ih} \cdot w_h & \text{if } e \in F \end{cases} \tag{3.13}$$

where $w_v$ is the weight of user $E$ node $v$, and $w_h$ is the weight of feature $E$ node $h$.

## 3.3 Learning the model with BPR

As mentioned in the previous section, our model relies on the BPR optimization algorithm to learn the weights. In this section, we will provide a detailed description of the updating process.

### 3.3.1 The sampling process

The training data used by BPR is made of $< u, i, j >$ samples where $u$ is the sampled user, $i$, an item rated by $u$, is the positive sample and $j$, an item not rated by $u$, is the negative one.
We select the negative sample for a user $u$ only among the items reachable with a three steps walk. This is because those are the item that will be considered for recommendation, and focusing on them may lead to faster training process.
As regards the order and the number of samples, we decided to abandon the idea of full cycles through the data, and to use a bootstrap sampling approach with replacement, as suggested in [43], because stopping can be performed at any step. Furthermore, traversing the data item-wise or user-wise will lead to poor convergence as there are so many consecutive updates on the same user-item pair [43]. As it can be observed in figure 3.5, extracted from the original paper, bootstrap sampling leads to a much faster convergence.

### 3.3.2 The updating process

BPR optimization algorithm relies on stochastic gradient descent to update the parameters of the model. Thus, the updating process is repeated for

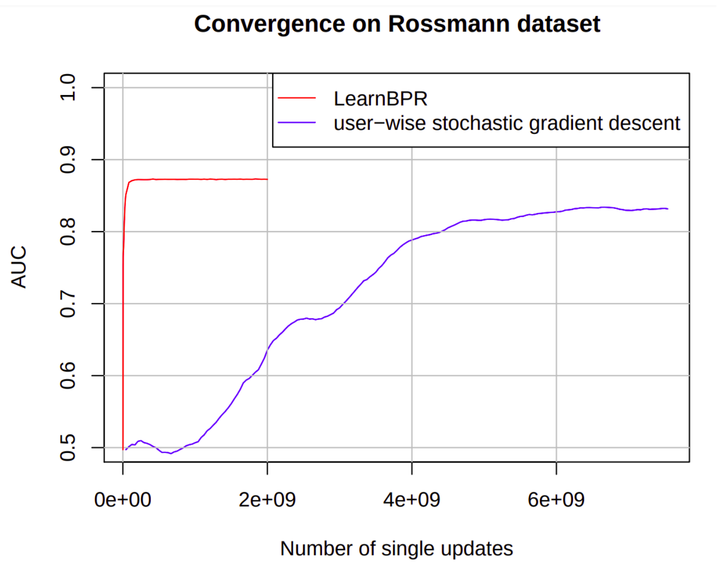


Figure 3.5: Comparison of BPR convergence with user-wise and bootstrap sampling(LearnBPR)

every sample $< u, i, j >$ with the following rule:

$$\Theta = \Theta + \alpha \left( \frac{1}{1 + e^{-x_{u,ij}}} \cdot \frac{\partial x_{u,ij}}{\partial \Theta} + \lambda \Theta \right) \tag{3.14}$$

As defined in the previous section, the score of user $u$ for item $i$ can be written as:

$$x_{ui} = P_{1,u} \odot P_2 \odot P_{3,i} \tag{3.15}$$

The relationship between user $u$ and the items $i$ and $j$, $x_{u,ij}$, is computed as the difference between user $u$'s scores of items $i$ and $j$.

$$\begin{aligned} x_{u,ij} &= P_{1,u} \odot P_2 \odot P_{3,i} - P_{1,u} \odot P_2 \odot P_{3,j} \\ &= P_{1,u} \odot P_2 \odot (P_{3,i} - P_{3,j}) \end{aligned} \tag{3.16}$$

By unpacking the scalar products we can write $x_{u,ij}$ as:

$$x_{u,ij} = \sum_{k \in I_u} P_{1,uk} \cdot \sum_{e \in E} P_{2,ke} \cdot (P_{3,ei} - P_{3,ej}) \tag{3.17}$$

Note that the second sum can be simplified. Instead of iterating over all $e \in E$, because of the high sparsity of the matrices, it is more efficient to do it only for the $e$ for which $P_{3,ei} - P_{3,ej} \neq 0$. From the point of view of the graph, this translates to considering only the $E$ nodes that are connected to item nodes $i$ or $j$, but not to both. If an $E$ node $e$ is connected to both $i$ and $j$, $P_{3,ei}$ is equal to $P_{3,ej}$ because the rows of $P_3$ represent uniform distributions. For this purpose, let's define $D_{ij}$ as the set of $E$ nodes connected to $i$ or $j$, but not both. We can write that

$$D_{ij} = \{e \in E | P_{3,ei} - P_{3,ej} \neq 0\} \tag{3.18}$$

By using this definition, we can write $x_{u,ij}$ as:

$$x_{u,ij} = \sum_{k \in I_u} P_{1,uk} \cdot \sum_{e \in D_{ij}} P_{2,ke} \cdot (P_{3,ei} - P_{3,ej}) \tag{3.19}$$

Given that we have to differentiate with respect to the weights used in the definition of $P_2$, both $P_{1,uk}$ and $(P_{3,ei} - P_{3,ej})$ are constant. Therefore, the derivative of the previous formula with respect to the weight vector $w$ can be computed as:

$$\frac{\partial x_{u,ij}}{\partial w} = \sum_{k \in I_u} P_{1,uk} \cdot \sum_{e \in D_{ij}} \frac{\partial P_{2,ke}}{\partial w} \cdot (P_{3,ei} - P_{3,ej}) \tag{3.20}$$

Since the elements of $w$, appear in its denominator of $P_{2,ke}$, the derivative is not immediate. We will have to differentiate with respect to the single weights. Thus, we can write the derivative of $x_{u,ij}$ with respect to the weight $w_f$ as:

$$\frac{\partial x_{u,ij}}{\partial w_f} = \sum_{k \in I_u} P_{1,uk} \cdot \sum_{e \in D_{ij}} \frac{\partial P_{2,ke}}{\partial w_f} \cdot (P_{3,ei} - P_{3,ej}) \tag{3.21}$$

For the sake of simplicity, let's differentiate $P_{2,ke}$ separately.

$$\frac{\partial P_{2,ke}}{\partial w_f} = \frac{\partial}{\partial w_f} \frac{T_{ke} \cdot w_e}{g_{ke}} \tag{3.22}$$

After applying the quotient rule, we obtain:

$$\frac{\partial P_{2,ke}}{\partial w_f} = \begin{cases} \frac{T_{ke} \cdot g_{ke} - T_{ke} \cdot w_e \cdot T_{kf}}{g_{ke}^2} & \text{if } f = e \\ -\frac{T_{ke} \cdot w_e \cdot T_{kf}}{g_{ke}^2} & \text{otherwise} \end{cases} \tag{3.23}$$

The derivative can be written as the sum of two components.

$$\frac{\partial P_{2,ke}}{\partial w_f} = \frac{T_{ke} \cdot g_{ke}}{g_{ke}^2} \cdot is_f(e) + \frac{-T_{ke} \cdot w_e \cdot T_{kf}}{g_{ke}^2} \tag{3.24}$$

where $is_f(e)$ is 1 if $f = e$, 0 otherwise.
Now we can write the derivative of $x_{u,ij}$ with respect to $w_f$ as $X + Y$, where $X$ contains the first term of the previous formula, while $Y$ contains the other one.
We can write $X$ as:

$$X = \sum_{k \in I_u} P_{1,uk} \cdot \sum_{e \in D_{ij}} \frac{T_{ke} \cdot g_{ke}}{g_{ke}^2} \cdot is_f(e) \cdot (P_{3,ei} - P_{3,ej}) \tag{3.25}$$

Note that the second sum is no longer necessary given that $is_f(e)$ will be 0 in all iterations except one, if $f \in D_{ij}$, at most. So we can rewrite $X$ as

$$X = \begin{cases} \sum_{k \in I_u} P_{1,uk} \cdot \frac{T_{kf} \cdot g_{kf}}{g_{kf}^2} \cdot (P_{3,fi} - P_{3,fj}) & \text{if } f \in D_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{3.26}$$

$X$ will be 0 in the derivative with respect all weights $w_f$, except when the $E$ node $f$ is connected to item node $i$ or item node $j$, but not both.
$Y$, on the other hand, can be written as:

$$\begin{aligned} Y &= -\sum_{k \in I_u} P_{1,uk} \cdot \sum_{e \in D_{ij}} \frac{T_{ke} \cdot w_e \cdot T_{kf}}{g_{ke}^2} \cdot (P_{3,ei} - P_{3,ej}) \\ &= -\sum_{k \in I_u} P_{1,uk} \cdot T_{kf} \cdot \sum_{e \in D_{ij}} \frac{T_{ke} \cdot w_e}{g_{ke}^2} \cdot (P_{3,ei} - P_{3,ej}) \end{aligned} \tag{3.27}$$

Since the second sum doesn't depend on the single weight for which we are differentiating, $w_f$ in this case, it can be precomputed to speed up the

updating process. In fact, with these optimizations on $X$ and $Y$, we observed a $\times 25$ speed up of the algorithm, which made the training phase faster and more scalable.
Finally, we can write the derivative of $x_{u,ij}$ with respect to the weight $w_f$ as:

$$\frac{\partial x_{u,ij}}{\partial w_f} = X + Y \tag{3.28}$$

The update of weight $w_f$ can be performed with:

$$w_{f,t+1} = w_{f,t} + \alpha \left( \frac{1}{1 + e^{-x_{u,ij}}} \cdot (X + Y) + \lambda \cdot w_{f,t} \right) \qquad \forall f \in E \tag{3.29}$$

## 3.4 Implementation detail

We implemented the model in python [35]. This choice is motivated by its large number of mathematical libraries. In particular, we used Numpy [52] and Scipy [21]. They provide data structures for efficient representations of large matrices, arrays and mathematical functions.
We used Cython [7] to implement the weights updating process. It allowed us to implement the most computationally expensive part of the algorithm directly in C [45], where we were able to use OpenMP [51] to easily parallelize the single weight updates.
One problem of the implementation of the described model is its time complexity. In order to update a single parameter, we need to iterate over the items of the sampled user. So a single parameter update has a time complexity of $\mathcal{O}(I)$, where $I$ is the total number of items. Since we have a parameter for every user and for every feature, the time complexity to handle a single training sample is of $\mathcal{O}(I \cdot (U + F))$, where $U$ is the number of users and $F$ is the number of features.
Note that without the optimizations described in the previous section, the time complexity required to handle a single training sample would have been $\mathcal{O}(I \cdot (U + F)^2)$.

# Chapter 4

# Results

## 4.1 Datasets

To test our model, we needed datasets that have both collaborative and content information. For this reason, we selected the Yahoo Movies dataset, the LastFm Hetrec dataset and the MovieLens Hetrec dataset. In this section, we will present the characteristics of the datasets and show how they were used for training and testing. In all three datasets, we transformed the collaborative feedback in implicit ratings, because, as already mentioned, implicit feedback are more common thanks to their easiness to create and their performance similar to the ones of explicit feedback.

### 4.1.1 LastFm HetRec

This dataset contains music artists listening information from LastFm online music system. It was released during the second International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011) at the fifth ACM Conference on Recommender Systems (RecSys 2011).
The collaborative part consists of the number of times each user has

listened to a given artist. There are 92834 listening count between its 1892 users and 17632 artists, which are the items in this case. To transform the collaborative data into implicit feedback, we converted all the listening counts into unary ratings. Thus, from the point of view of the recommender, there is no difference if a user listened to a given artist just one or multiple times. We decided to not remove the listening counts below a certain threshold to avoid reducing the amount of feedback, which is already very limited. Moreover, a user listening to an artist only once does not necessarily mean that they don't like them.
From the distribution of feedback with respect to the items, displayed in figure 4.1, we can observe that a great part of the interactions belong to the first items, which are the most popular ones. This, referred to as long tail, is very common since popular items are more likely to be interacted with.
The content information, on the other hand, is made of descriptive tags users have assigned to the artists. There are 11946 distinct tags assigned 186479 times to the artists. In this case, there was no need perform any kind of preprocessing on the tags. They had numerical ids that could be used directly to create the ICM. The latter is a binary matrix that can be obtained by simply using one-hot encoding on the tags.
The distribution of the number of features per item, ordered by item popularity, is displayed in figure 4.2. Also in this case we can note the long tail effect, although not as pronounced as with the number of feedback distribution. This is due to the fact that the descriptions of items are provided by users through tags; popular items are subject to more tagging.

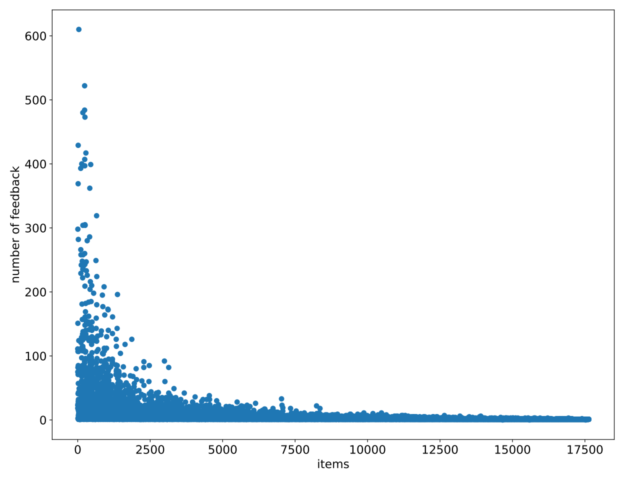


Figure 4.1: Number of feedback per item in the the LastFm HetRec dataset

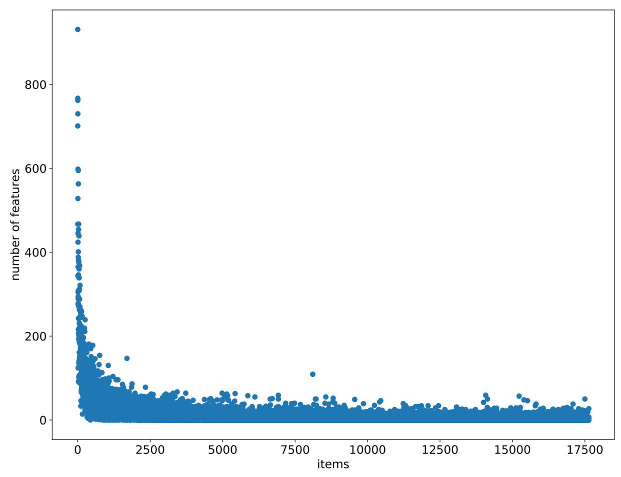


Figure 4.2: Number of features per item in the LastFm HetRec dataset, ordered by item popularity

### 4.1.2 Yahoo Movies

This dataset contains a sample of the Yahoo Movies community's preferences for various movies. It also has a large amount of descriptive information about the movies, including cast, crew, synopsis, genre, average ratings, awards, etc.
The collaborative part of the dataset is made of 221364 ratings, from 1 to 5, given by the 7642 users to the 11916 items.

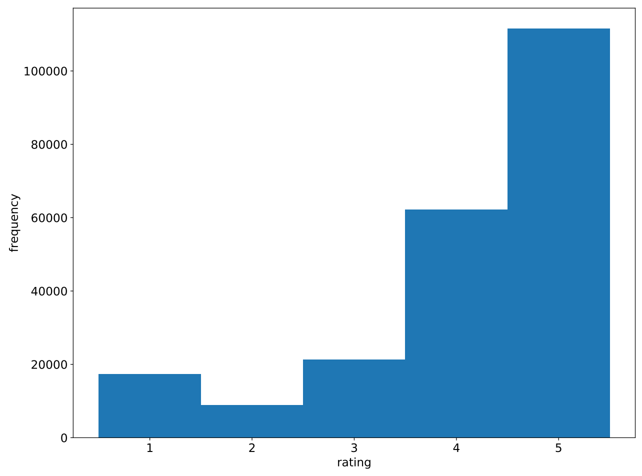


Figure 4.3: Ratings frequency in the Yahoo Movies dataset

The frequency of the different rating values can be observed in figure 4.3. Unlike with the LastFm Hetrec dataset, in this case, we have the concept of negative feedback, which corresponds to a low rating. Thus, we converted the ratings into implicit feedback by removing the ones below 3. We decided to include the ratings equal to 3 as positive feedback to avoid removing too much data. With this operation, the number of interactions decreased by 12%. The distribution of the remaining 195041 feedback with respect to the items can be observed in 4.4.
As mentioned before, the dataset has a large amount of item descriptive

data. Most of them can be directly used without any preprocessing. For instance the actors, crew members and directors are represented with unique ids, while the distributor, list of genres and list of awards have unique string representations. The title and the synopsis need more work on the other hand. With titles, the non-alphanumerical characters are removed before tokenizing over spaces. From the obtained tokens, stop words are discarded and the rest are used as features. The preprocessing of the synopsis text is similar, but the token are stemmed before being discarded or used. Also in this case the ICM is obtained by using one-hot encoding over the set of features. It has 221433 columns, one for each feature, and 750234 non-zero cells, one for each item-feature relation. The distribution of the features with respect to the items is displayed in figure 4.5. As it can be observed, in this case, there is no long tail because the content information is not generated by users.

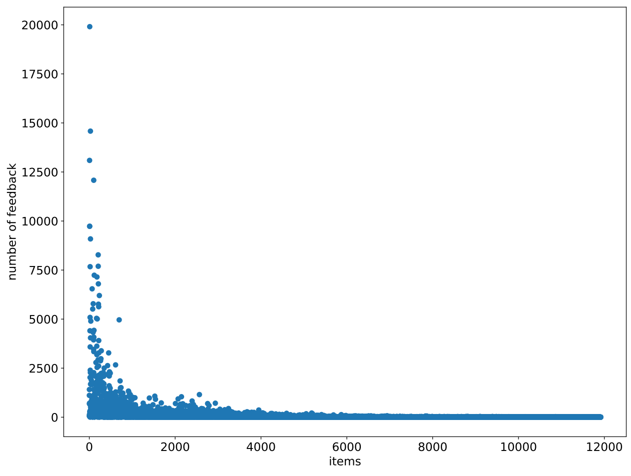


Figure 4.4: Number of feedback per item in the the Yahoo Movies dataset

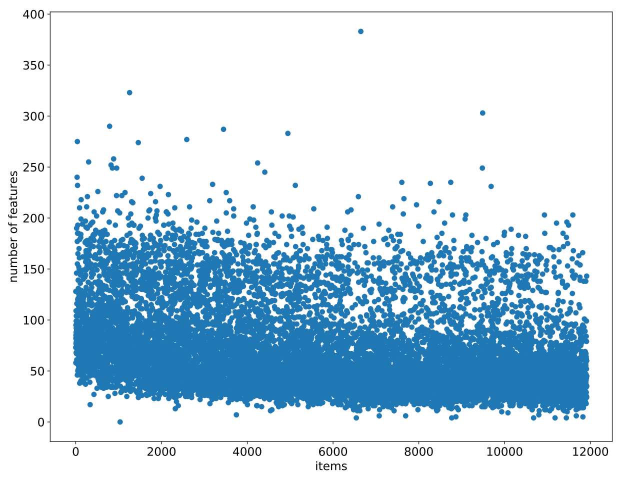


Figure 4.5: Number of features per item in the Yahoo Movies dataset, ordered by item popularity

### 4.1.3 MovieLens HetRec

This dataset is a subset of the popular MovieLens10M dataset that has been enriched by linking the movies with their corresponding web pages at Internet Movie Database (IMDb) and Rotten Tomatoes movie review systems. It was released during the second International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011) at the fifth ACM Conference on Recommender Systems (RecSys 2011).

It has 855598 ratings, from 1 to 5, given by its 2113 users to its 10197 movies. Their distribution can be observed in figure 4.6. Also in this case we transformed the explicit feedback into implicit ones by removing those below 3, and considering the remaining ones as positive interactions. This operation removed about 17% of the available feedback.

The content data contains descriptions of movies and 13222 tags assigned 47957 times by users. Among the descriptions available, besides the tags,

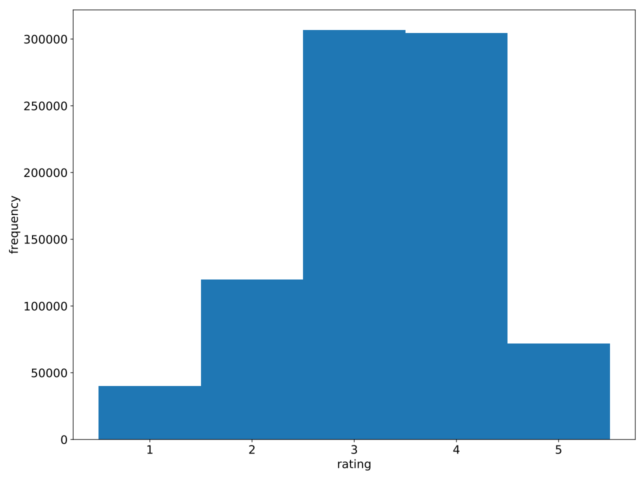


Figure 4.6: Ratings frequency in the MovieLens HetRec dataset

we used the list of genres, the list of directors and the list of actors. Tags, directors and actors are represented by unique ids, so transforming them into features didn't require any preprocessing. From this we obtained 104698 features with 314501 item-feature relations.
Given that the amount of data is still significantly larger than the two previous datasets, we decided to clean further the data and remove the potentially less relevant part. We removed users, items and features that have a low number of interactions. In particular, we removed from the URM the users and the items with less than 5 interactions. From the ICM, we also removed the items with less than 5 features and the features with less than 5 items. These two operations were recursively repeated until every row and every column of the URM and of the ICM had at least 5 elements. This reduced the number of items by 32%, while cutting the number of feedback in the URM by only 2%. Their distribution with respect to the items is displayed in figure 4.7.
For the content information, on the hand, the number of features decreased by almost 90%, while the number of item-feature relations was reduced by

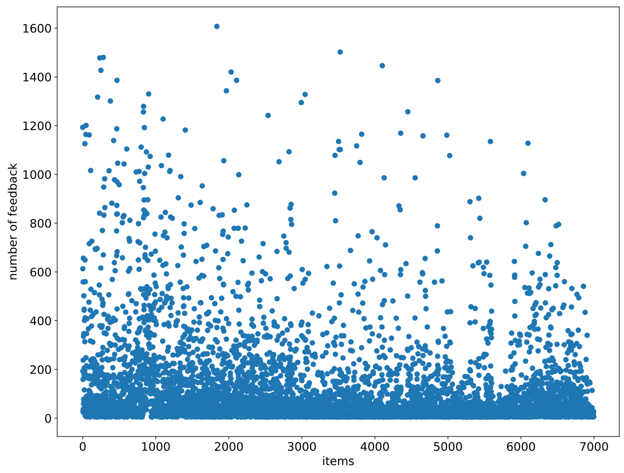


Figure 4.7: Number of feedback per item in the the Movies dataset

58%. From their distribution with respect to the items displayed in figure 4.8, we can observe that popular items have a higher number of features. This is due to the fact that part of the content data is generated by users through tags.

## 4.1.4 Splitting procedure

The purpose of this thesis is to design a new hybrid recommendation model that is able to combine the advantages of collaborative algorithms with the ones of content algorithms. Thus, the experiments were carried out in two different configurations: warm and cold start scenarios. In both cases, the interactions, obtained after performing the operations described in the relative dataset section, are split into three part: training set, validation set and testing set. The first two parts are used to train the model and find the optimal hyperparameters, while the testing set is used for the final evaluation of the algorithms.
In a first moment, the data is split into training set and testing set, then

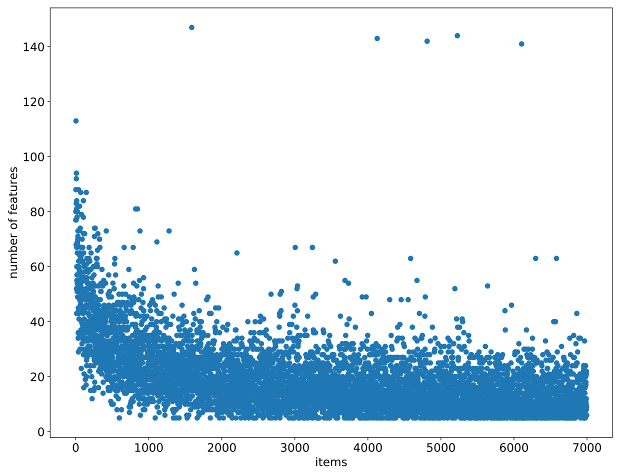


Figure 4.8: Number of features per item in the MovieLens HetRec dataset, ordered by item popularity

the training set is split again into a new training set and validation set. This last splitting procedure is the same in both configurations. It consists of randomly selecting 80% of the training set data, obtained with the first split, as the new training set and the remaining 20% as the validation set. The difference between the two configurations lies in how the first split is performed.

- **Warm start**: in this case, the first split is similar to the second one already described. It is done by randomly selecting 20% of the training interactions as testing set and the remaining 80% as the new training set.

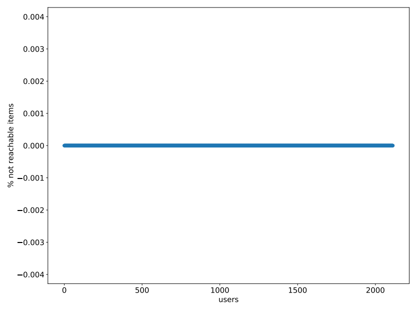
- **Cold start**: this configuration is used to simulate a cold start scenario. Its purpose is to hide the items we wish to perform evaluation on from the training process, to make them look new during the tests. In order to do so, we randomly select 20% of the items and used their interactions as testing set. In this way, the training set and validation set, which are made of the interactions of the remaining 80% of items, doesn't contain any feedback of the cold items.

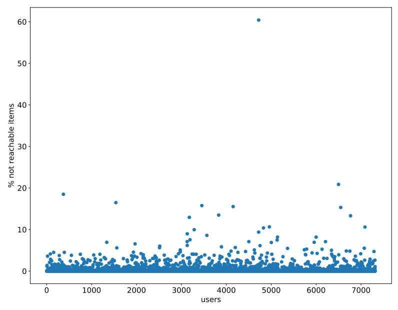
## 4.2 Experiments

In this section, we will describe the changes applied to the model to improve the performance and discuss the obtained results.

### 4.2.1 Evaluating the sampling process

As described in section 3.3.1, we tried to select the negative sample for a sampled user $u$ only among the items reachable with a three steps walk, since those are items that will be considered during the recommendation phase. However, the model behaves in the same way if the negative sample is randomly selected among all items, without filtering those non reachable with three steps walks. As it can be observed from figure 4.9, this is due to the fact that in the used datasets, a user can reach almost any item with a three steps walk.


(a) Hetrec MovieLens


(b) Yahoo Movies

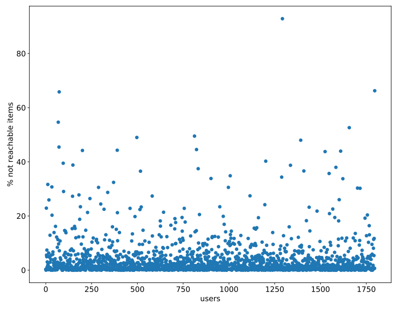

(c) Hetrec LastFm

Figure 4.9: Percentage of items non reachable in a 3 steps walk

Given that in this situation the computation cost of the filtering process is not justified by the results, we decided to select the negative sample randomly from all the items. However, in larger and more sparse datasets, it can make an impact.

## 4.2.2 Simplified gradient

The model presented in this thesis relies on stochastic gradient descent to learn the weights of the model. This means that for every training sample an update is performed with the process described in section 3.2. For a single weight $w_f$, we can write its updating formula as:

$$w_{f,t+1} = w_{f,t} + \alpha \left( \frac{1}{1+e^{-x_{u,ij}}} \cdot (X+Y) + \lambda \cdot w_{f,t} \right) \tag{4.1}$$

where

$$X = \begin{cases} \sum_{k \in I_u} P_{1,uk} \cdot \frac{T_{kf} \cdot g_{kf}}{g_{kf}^2} \cdot (P_{3,fi} - P_{3,fj}) & \text{if } f \in IJ \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

and

$$Y = -\sum_{k \in I_u} P_{1,uk} \cdot T_{kf} \cdot \sum_{e \in D_{ij}} \frac{T_{ke} \cdot w_e}{g_{ke}^2} \cdot (P_{3,ei} - P_{3,ej}) \tag{4.3}$$

In both cases, the denominator can be computed with the following formula:

$$g_{ie} = \begin{cases} \sum_{v \in U} R_{vi} \cdot w_v & \text{if } e \in U \\ \sum_{h \in F} C_{ih} \cdot w_h & \text{if } e \in F \end{cases} \tag{4.4}$$

Note that, since we are using implicit feedback, both the URM and the ICM, respectively represented by $R$ and $C$ in the previous formula, are binary matrices. Thus, in practice, $g_{ie}$ is just a sum of weights: the weights of the user nodes connected to the item node $i$, if $e$ is a user node, or the weights of the feature nodes connected to $i$, if $e$ is a feature node.
This property can become a drawback of the updating process. The purpose of the latter is to learn the weights of the $E$ nodes. At the beginning of the learning phase, all nodes have the same initial weight. Progressively, based on their relevance, some weights will increase, while other decrease. The differences of values between the relevant weights and the not relevant ones will become larger as the learning process advances. Therefore, an item $i$ only connected to $E$ nodes with relatively low weights will have a relatively low $g_{ie}$. Given that we have $g_{ie}^2$ in the denominator of both $X$ and $Y$, the contribution of item $i$ can largely outweigh those of the other items.
To make clear the reason why this can have a negative effect on the learning procedure, let's make an example. Imagine to have a user sample

$u$ with several items, including item $h$ which has an extremely low $g_{he}$. From the formula of $Y$, we can note that the value of the inner sum is computed for every item $k$ of $u$. In this case, though, the value of the inner sum relative to item $h$ will be much higher than the other values because of the very low $g_{he}$. Moreover, the square at the denominator will increase further the difference. In extreme cases, the value of item $h$ can make the values of the other items neglectable. This would lead to two problems: the update step would be too large, reducing the ability of the model to find the optimal convergence, and the other items of $u$ would not affect it in any way.
To reduce the magnitude of this problem, we simplified the gradient by removing the square at the denominator. The formula of $X$ and $Y$ become:

$$X = \begin{cases} \sum_{k \in I_u} P_{1,uk} \cdot \frac{T_{kf} \cdot g_{kf}}{g_{kf}} \cdot (P_{3,fi} - P_{3,fj}) & \text{if } f \in IJ \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

and

$$Y = -\sum_{k \in I_u} P_{1,uk} \cdot T_{kf} \cdot \sum_{e \in D_{ij}} \frac{T_{ke} \cdot w_e}{g_{ke}} \cdot (P_{3,ei} - P_{3,ej}) \tag{4.6}$$

The difference of performance between the original gradient and the simplified one is displayed in figure 4.10. We observed an improvement of about 3% on the test set of LastFm HetRec.

The experiments that will be presented in the rest of this section were executed on the model with the simplified gradient.

### 4.2.3 A popularity based re-ranking

It has been demonstrated that the ranking of items based on the transition probability matrix $P^3$ is strongly influenced by the popularity of items [11]. This is a common problem in recommender systems. many collaborative filtering algorithms are biased toward popular items, which leads to a lower recommendation diversity.

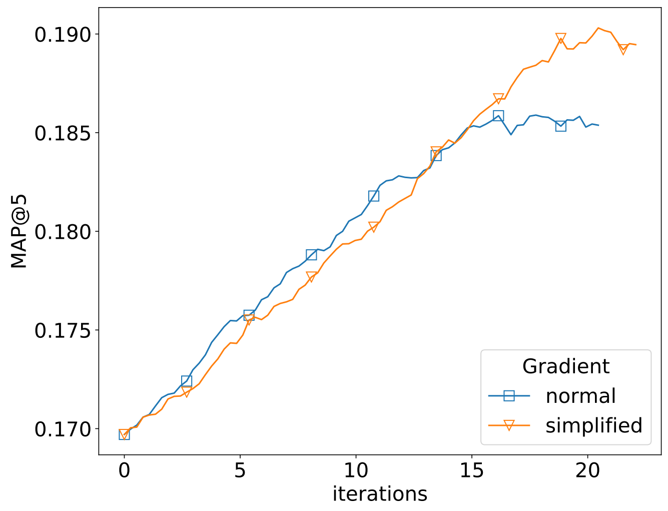


Figure 4.10: Evaluation on the test set of the LastFm HetRec dataset in a warm start configuration with the normal gradient and the simplified one

Even though our model doesn't rank items based on the probability matrix $P^3$, it suffers from the same problem. In fact, in more general terms, this issue affect ranking procedures that rely on discrete uniform distributions. To explain how our model is affected, let's consider the following example. Imagine to have a user $u$ and two items $i$ and $j$, with $j$ much more popular than $i$. The score of user $u$ to item $i$ is computed as the sum of the probabilities to reach the $E$ node $e$ from $u$ multiplied by the probability to reach $i$ from $e$, for every $e$ connected to $i$. Let's consider a case in which $i$ is only connected to a few number of $E$ nodes that can be reached from $u$ with high probabilities, while $j$, being more popular, is connected to a great number of $E$ nodes, but their probabilities to be reached from $u$ are very low. If the indegree difference between node $j$ and node $i$ is large enough, the model will rank $j$ above $i$ because the sum of the many small probabilities will be higher that the one of few large probabilities.
To solve this problem, we decided to adapt a re-ranking procedure similar

to the one employed by $RP^3_\beta$. In this case, however, the popularity penalty is computed from the both collaborative and content data. By using only the former, in cases in which some items don't have many features, if content data is provided through tags for example, they would be disadvantaged.
Therefore, the score of user $u$ to item $i$ that was defined as

$$x_{ui} = P_{1,u} \times P_2 \times P_{3,i} \tag{4.7}$$

becomes

$$x_{ui} = \frac{P_{1,u} \odot P_2 \odot P_{3,i}}{d_i^\beta} \tag{4.8}$$

where $d_i$ is the number of $E$ nodes connected to the item node $i$, and $\beta$ can be used to control to magnitude of the popularity penalty. A value of 0 means no penalty, while highers values correspond to higher penalties. A comparison of the performance of the model with and without popularity penalty can be observed in figure 4.11. With a the parameter $\beta$ equal to 0.15, we obtained an improvement of about 2.5% on the test set of the LastFm HetRec dataset in a warm start configuration.

### 4.2.4 A mini-batch based update

The BPR algorithm used to learn the parameters of the model rely on stochastic gradient descent, which is a stochastic approximation of the full gradient descent algorithm. With the latter, to perform an update, the gradient of the cost function is computed over all the samples in the training set. While this approach produce a much more stable learning process, it is very computationally expensive. In cases of large datasets, using full gradient descent is unfeasible. Stochastic gradient descent simplify the updating process by computing the gradient over only on a single training sample. This approximation leads to a noisy, but much faster, learning process since all the parameters of the model are updated based on a single sample.

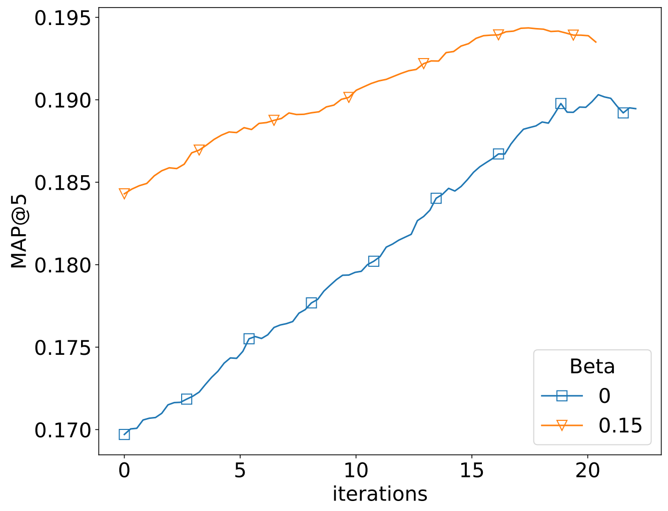


Figure 4.11: Performance on the test set of the LastFm HetRec dataset with different values of $\beta$ in a warm start scenario

Mini-batch gradient descent is a compromise between the two approaches; it tries to find a balance between the robustness of full gradient descent and the efficiency of the stochastic approximation. The basic idea is to compute the gradient over $n$ samples, instead of just one. The parameter $n$ is the size of the mini-batches and can be used to control the degree of approximation of the gradient. A value of 1 corresponds to the stochastic version, while a value equal to the number of training samples corresponds to the full gradient.
By using mini-batches of size $n$, the update rule

$$\Theta = \Theta + \alpha \left( \frac{1}{1 + e^{-x_{u,ij}}} \cdot \frac{\partial x_{u,ij}}{\partial \Theta} + \lambda \Theta \right) \tag{4.9}$$

becomes

$$\Theta = \Theta + \alpha \frac{\phi_n}{n} \tag{4.10}$$

where

$$\phi_n = \sum_{b=1}^{n} \frac{1}{1 + e^{-x_{u_b,ij_b}}} \cdot \frac{\partial x_{u_b,ij_b}}{\partial \Theta} + \lambda\Theta \tag{4.11}$$

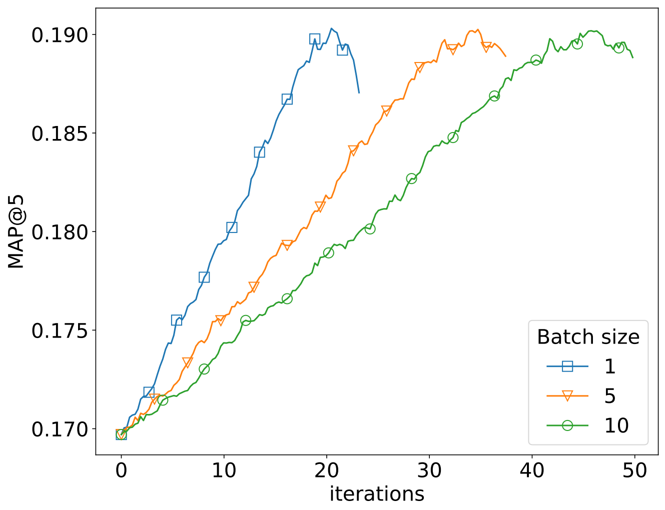


Figure 4.12: Performance on the test set of LastFm HetRec dataset with different mini-batch sizes in a warm start scenario

As it can be observed in figure 4.12, the mini-batch implementation did not improve the performance of the system. This may be due to the fact that the updates of the weights are very sparse. However, with a higher batch size the model is more stable, thus finding the peak becomes easier since it doesn't start to overfit immediately. For this reason we decided to use a batch size of 5.

## 4.3 Comparing the model with the state of art

In this section, we compare the performance of our model with several state of the art algorithms in both warm and cold start configurations. In particular we used the following algorithms:

- **Item KNN CBF**: a content based filtering based on k nearest neighbors
- **Item KNN CF**: an item based collaborative filtering algorithm based on k nearest neighbors
- $\mathbf{P^3_\alpha}$: the graph based algorithm described in section 2.6.1
- $\mathbf{RP^3_\beta}$: the graph based algorithm described in section 2.6.2
- **CF-CBF Hybrid**: a weighted hybrid between item based collaborative filtering and content based filtering based on k nearest neighbors. The algorithm described in section 2.7.1
- **GHR**: Graph-based Hybrid Ranking is the model proposed in this thesis, in section 3.2.

In the rest of this section we will present the results of these algorithms on the test sets of the three datasets already presented.

### 4.3.1 Warm start

In this section we analyze the performance @5 and @10 of the different algorithms in a warm start scenario in the three datasets. The algorithms are divided into three categories: content, collaborative and hybrid. As regards the evaluation metrics, we use two classification metrics, precision and recall, two rank accuracy metrics, MAP and NDCG, and a beyond accuracy metric, average popularity.

**LastFm HetRec**

The results in a warm start configuration are displayed in tables 4.1 and 4.2 for performance @5 and @10, respectively. As expected, the collaborative algorithms have higher performance than the content one, and the hybrids outperform both classes. However, one of the problem of collaborative algorithms is that they emphasize popular items over those from the long tail. In fact, as it can be observed from the two tables, they have a higher average popularity.
Our model outperformed the pure collaborative algorithms and the hybrid one in all the metrics. Moreover, it was also able to achieve a lower average popularity. This means that, unlike the CF-CBF hybrid, which has an average popularity similar to the ones of pure collaborative algorithms, our model is able to use the content information to decrease the bias toward popular items of collaborative algorithms.

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCBF | 0.21672 | 0.11058 | 0.15551 | 0.15690 | **0.31004** |

(a) Content

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | 0.23663 | 0.12093 | 0.17853 | 0.17319 | 0.43961 |
| $P_\alpha^3$ | 0.21278 | 0.10938 | 0.15587 | 0.15435 | 0.53288 |
| $RP_\beta^3$ | 0.22556 | 0.11478 | 0.16978 | 0.16477 | 0.49265 |

(b) Collaborative

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| CF-CBF hybrid | 0.25037 | 0.12885 | 0.18890 | 0.18316 | 0.42343 |
| GHR | **0.26048** | **0.13495** | **0.19405** | **0.18899** | 0.35749 |

(c) Hybrid

Table 4.1: Performance @5 of different algorithm on the test set of LastFm HetRec dataset in a warm start configuration

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCBF | 0.16289 | 0.16632 | 0.10745 | 0.19543 | **0.26505** |

(a) Content

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | 0.17551 | 0.17844 | 0.12277 | 0.21333 | 0.38449 |
| $P^3_\alpha$ | 0.15895 | 0.16197 | 0.10727 | 0.19119 | 0.46465 |
| $RP^3_\beta$ | 0.16778 | 0.17088 | 0.11593 | 0.20387 | 0.42903 |

(b) Collaborative

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| CF-CBF hybrid | 0.18472 | 0.18843 | 0.13089 | 0.22498 | 0.36795 |
| GHR | **0.19249** | **0.19770** | **0.13493** | **0.23294** | 0.30690 |

(c) Hybrid

Table 4.2: Performance @10 of different algorithm on the test set of LastFm HetRec dataset in a warm start configuration

### Yahoo Movies

The results in a warm start configuration are displayed in tables 4.3 and 4.4 for the performance @5 and @10, respectively. In this case, the gap between collaborative and content algorithms is even larger. This is mainly due to the quality of the content data. Among the collaborative algorithms, the two graph based ones have the lowest performance. While $RP^3_\beta$ produces considerably better results than $P^3_\alpha$, its re-ranking procedure is not enough to achieve a better result than the item based CF algorithm. For both @5 and @10 results, our model slightly outperforms the collaborative algorithms and the CF-CBF hybrid in all the considered metrics. Even for the average popularity we don't have a gap between our model and CF-CBF the hybrid, as with the LastFm HetRec dataset. As previously mentioned, this may be due to the quality of the content data.

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCBF | 0.07301 | 0.08067 | 0.06007 | 0.08012 | **0.06360** |

(a) Content

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | 0.18262 | 0.23945 | 0.19251 | 0.23554 | 0.41918 |
| $P^3_\alpha$ | 0.15675 | 0.21173 | 0.15447 | 0.20007 | 0.56090 |
| $RP^3_\beta$ | 0.18176 | 0.23900 | 0.18583 | 0.23047 | 0.47191 |

(b) Collaborative

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| CF-CBF hybrid | 0.18690 | 0.24944 | 0.19848 | 0.24433 | 0.39359 |
| GHR | **0.19367** | **0.25053** | **0.20108** | **0.24663** | 0.37086 |

(c) Hybrid

Table 4.3: Performance @5 of different algorithm on the test set of Yahoo Movies dataset in a warm start configuration

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCBF | 0.05591 | 0.11650 | 0.06047 | 0.09886 | **0.05099** |

(a) Content

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | 0.13498 | 0.34318 | 0.19968 | 0.28532 | 0.34932 |
| $P^3_\alpha$ | 0.11936 | 0.30685 | 0.16346 | 0.24659 | 0.40389 |
| $RP^3_\beta$ | 0.13724 | 0.34774 | 0.19574 | 0.28304 | 0.38564 |

(b) Collaborative

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| CF-CBF hybrid | 0.13713 | 0.35381 | 0.20636 | 0.29453 | 0.33763 |
| GHR | **0.14267** | **0.36182** | **0.20797** | **0.29977** | 0.32649 |

(c) Hybrid

Table 4.4: Performance @10 of different algorithm on the test set of Yahoo Movies dataset in a warm start configuration

### 4.3.2 MovieLens HetRec

The results in a warm start configuration are displayed in tables 4.5 and 4.6 for the performance @5 and @10, respectively.
In this dataset the difference of performance between collaborative and content algorithms is very small. In fact, the CBF outperformed $P^3_\alpha$, and had similar results to $RP^3_\beta$. This may be thanks to the quality of the tags provided by users, which in many cases can help reduce the gap between content and collaborative algorithms.
An unexpected result is the fact that the CF-CBF hybrid was not able to produce better results than the item based CF algorithm. A possible explanation is that the collaborative information is very similar to the content one in this dataset, because of the tags provided by users, so by combining them, the system will not be able to learn anything new.

Our model, produced results similar to the ones of the item based CF algorithm. The latter had slightly better accuracy metrics values, while our model had slightly higher ranking metrics values and an average popularity similar to the one of the CBF algorithm.
The low values of the recall, and high values of the other metrics, are caused by the large number of items in the users' profiles.

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCBF | 0.48131 | 0.05428 | 0.41748 | 0.11555 | **0.69565** |

(a) Content

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | **0.50575** | **0.05556** | 0.43194 | 0.11529 | 0.80178 |
| $P^3_\alpha$ | 0.44422 | 0.04785 | 0.37292 | 0.10150 | 0.89698 |
| $RP^3_\beta$ | 0.46324 | 0.05062 | 0.39199 | 0.10651 | 0.88939 |

(b) Collaborative

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| CF-CBF hybrid | 0.49681 | 0.05446 | 0.43055 | 0.11583 | 0.82754 |
| GHR | 0.50461 | 0.05426 | **0.43799** | **0.11637** | 0.71697 |

(c) Hybrid

Table 4.5: Performance @5 of different algorithm on the test set of MovieLens HetRec dataset in a warm start configuration

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCBF | 0.40399 | 0.08632 | 0.31856 | 0.15075 | **0.65101** |

(a) Content

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | **0.44536** | **0.09165** | **0.35344** | 0.15632 | 0.77553 |
| $P^3_\alpha$ | 0.39225 | 0.08177 | 0.29875 | 0.13845 | 0.84848 |
| $RP^3_\beta$ | 0.40694 | 0.08434 | 0.31521 | 0.14386 | 0.84216 |

(b) Collaborative

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| CF-CBF hybrid | 0.42720 | 0.08971 | 0.34054 | 0.15475 | 0.79411 |
| GHR | 0.43352 | 0.09098 | 0.34541 | **0.15713** | 0.713233 |

(c) Hybrid

Table 4.6: Performance @10 of different algorithm on the test set of MovieLens HetRec dataset in a warm start configuration

### 4.3.3 Cold start

In this section we analyze the performance @5 of the content and hybrid algorithms in a cold start scenario. The pure collaborative algorithms are excluded since they are not able to provide recommendation in this configuration. As regards the evaluation metrics, we use the same as in the warm start configuration, except for the average popularity. The latter is computed from the popularity of items in the trainset. Since the item we are performing evaluation on are not present in the trainset, their average popularity will be 0 in this configuration.

#### LastFm HetRec

The performance @5 in cold start scenario are displayed in table 4.7.
As expected, the CBF algorithm produced better results than the CF-CBF

hybrid. Normally, the CF-CBF hybrid gives a higher weight to its collaborative component, since it provides better results in general. Therefore, in a cold start scenario, it is not able to fully exploit the capabilities of its content based component.
Anyway, they are both outperformed by our model in all the metrics.

| **Algorithm** | **Precision** | **Recall** | **MAP** | **NDCG** |
|---|---|---|---|---|
| ItemKNNCBF | 0.11774 | 0.05812 | 0.09009 | 0.08328 |
| CF-CBF hybrid | 0.11103 | 0.05470 | 0.08342 | 0.07783 |
| GHR | **0.13915** | **0.06794** | **0.10013** | **0.09425** |

Table 4.7: Performance @5 of different algorithm on the test set of LastFm HetRec dataset in a cold start configuration

### Yahoo Movies

In the cold start scenario, we obtained quite low results. As it can be observed in tab 4.8, our model did not outperform the pure content based algorithm, but it had very similar outcomes. The CF-CBF hybrid, on the other hand, is not able to provide competitive recommendation. Because of the enormous gap between the collaborative and the content recommenders, the collaborative part of the weighted hybrid has a much higher weight. Consequently, the hybrid is not able to take advantage of the content part in a cold start scenario. Our model, on the other hand, is still able to exploit the content information, and to produce results comparable to the ones of the content based algorithm.

| **Algorithm** | **Precision** | **Recall** | **MAP** | **NDCG** |
|---|---|---|---|---|
| ItemKNNCBF | **0.02738** | **0.03279** | **0.02248** | **0.03006** |
| CF-CBF hybrid | 0.00516 | 0.00474 | 0.00500 | 0.00533 |
| GHR | 0.02319 | 0.02376 | 0.01901 | 0.024383 |

Table 4.8: Performance @5 of different algorithm on the test set of Yahoo Movies dataset in a cold start configuration

#### MovieLens HetRec

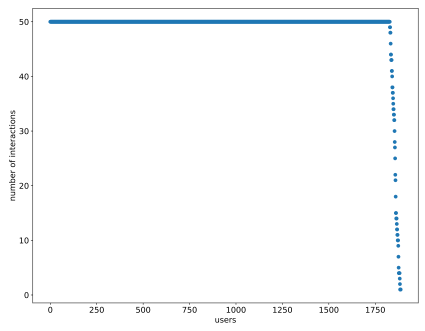
The performance @5 are displayed in table 4.9.
Also in this case, the proposed model had performance similar to those of the CBF, although slightly lower. The CF-CBF hybrid, on the other hand, was able to only reach half of their performance.

| **Algorithm** | **Precision** | **Recall** | **MAP** | **NDCG** |
|---|---|---|---|---|
| ItemKNNCBF | **0.68664** | **0.08150** | **0.63700** | **0.16424** |
| CF-CBF hybrid | 0.38559 | 0.04176 | 0.28672 | 0.08361 |
| GHR | 0.64540 | 0.07344 | 0.59013 | 0.15002 |

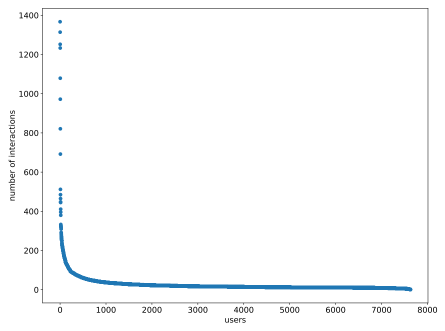
Table 4.9: Performance @5 of different algorithm on the test set of MovieLens HetRec dataset in a cold start configuration

## 4.4 The effect of the lengths of users' profiles

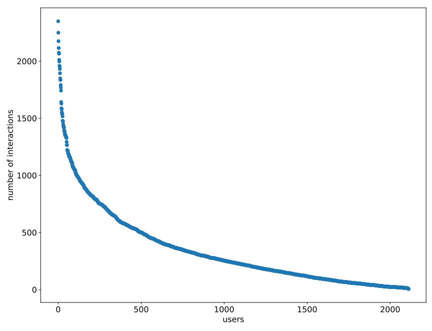
In most cases, collaborative algorithms are able to provide better recommendations than content based ones, and they also produce more diverse recommendations [6]. They are based on users' behaviors, so, in order to perform at their best, they require enough data from the users. As it can be observed from figure 4.13, in some cases, users can have short profiles, with a low number of interactions.
In this section, we analyze how the short lengths of users' profiles affect



(a) LastFm HetRec



(b) Yahoo Movies



(c) MovieLens HetRec

Figure 4.13: Length of users' profiles in the datasets

our model. As it can be observed from figure 4.13, in both LastFm HetRec and MovieLens HetRec datasets there are not many users with very low number of interactions. On the Yahoo Movies dataset, on the other hand, the presence of the long tail indicates that most users have a very short profile. Actually, the long tail is present in almost every real dataset. Its absence in the LastFm HetRec and MovieLens HetRec datasets is due to the subsampling performed to create them.
Anyway, we decided to test the collaborative and hybrid algorithms on the Yahoo Movies dataset ignoring the users that in the trainset have less

than:

- 10 interactions (28% of the users)
- 20 interactions (75% of the users)
- 30 interactions (85% of the users)

Just for comparison, in the MovieLens HetRec dataset, only 9% of the users have less than 30 interactions in the trainset, while in the LastFm dataset almost all users have 50 interactions.
The results for the three configurations are displayed in tables 4.10, 4.11 and 4.12, respectively.

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | 4% | −7% | 0% | −4% | **-5%** |
| CF-CBF hybrid | 3% | −8% | −1% | −4% | **-5%** |
| $P^3_\alpha$ | 2% | −9% | −4% | −7% | −1% |
| $RP^3_\beta$ | 4% | −8% | −1% | −5% | −3% |
| GHR | **6%** | **-6%** | **1%** | **-2%** | −3% |

Table 4.10: Variation of performance @5 on the test set of Yahoo Movies in warm start without users with less than 10 interactions with respect to evaluation on all users

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | 42% | −33% | 10% | −14% | **-25%** |
| CF-CBF hybrid | 37% | −36% | 5% | −17% | **-25%** |
| $P^3_\alpha$ | 29% | −40% | −6% | −24% | −1% |
| $RP^3_\beta$ | 41% | −34% | 8% | −15% | −11% |
| GHR | **48%** | **-29%** | **15%** | **-10%** | −11% |

Table 4.11: Variation of performance @5 on the test set of Yahoo Movies in warm start without users with less than 20 interactions with respect to evaluation on all users

| **Algorithm** | **Prec.** | **Rec.** | **MAP** | **NDCG** | **Avg. Pop.** |
|---|---|---|---|---|---|
| ItemKNNCF | 67% | −48% | 25% | −22% | −38% |
| CF-CBF hybrid | 59% | −51% | 16% | −26% | **-39%** |
| $P^3_\alpha$ | 46% | −55% | 2% | −34% | −2% |
| $RP^3_\beta$ | 65% | −48% | 22% | −24% | −16% |
| GHR | **72%** | **-45%** | **30%** | **-19%** | −16% |

Table 4.12: Variation of performance @5 on the test set of The Yahoo Movies in warm start without users with less than 30 interactions with respect to evaluation on all users

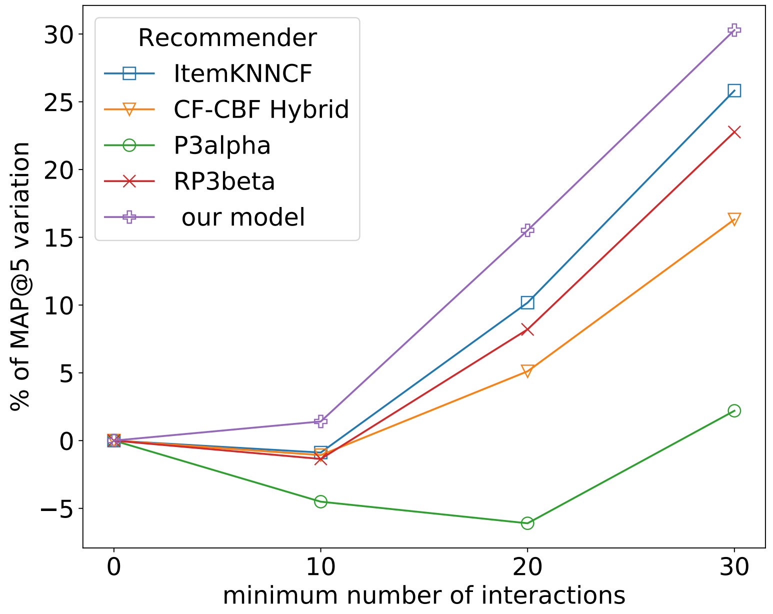


Figure 4.14: Variation of MAP@5 without users with short profiles on the test set of The Yahoo Movies dataset in warm start with respect to MAP@5 on all users. The value on the X axis represents the minimum number of interactions required for a user to be included in the evaluation

Note that the recall values decrease progressively because the average length of users' profiles is increasing.
To help visualize the results, the variations relative to the MAP are displayed in figure 4.14. As expected, as the average length of the profiles increases, the performance of the considered algorithms improve. Our model, in particular, increases at a higher rate. Therefore we can deduce that, with respect to the other collaborative and hybrid algorithms, our model performs better with users with a long profile; it is able to better exploit the amount of information available. In fact, we obtained the best results in the LastFm HetRec dataset, which doesn't have users with short profiles. However, since the MAP is averaged over all users, this means that for the users we ignored during the evaluation, the MAP of our model decreases also at a higher rate.

# Chapter 5

# Conclusion

In this thesis, we proposed a new recommender system that combines collaborative and content information into a new graph-based hybrid model. Therefore, in the graph, we have three types of nodes, user, item and feature nodes, and two types of paths that, from a user node, can lead to an item node with a three steps walk: the collaborative path, which passes through another user node, and the content path, which passes through a feature node. We merged the two kind of paths into a single one, and used BPR to the learn the weights of user and feature nodes, which are proportional the their probabilities of being reached from an item node. We compared our model with several state of art approaches of different types, collaborative, content and hybrid recommenders, in both warm and cold start scenarios. We also analyzed the performance of the proposed model on users with different profile lengths.
In a warm start scenario, the collaborative algorithms produced better results than the content one, as we expected, and the hybrids had higher performance than both classes. In most cases, our model outperformed the collaborative algorithms, and had slightly better results than the CF-CBF weighted hybrid. Moreover, it was able to achieve an average popularity score closer to the one of the content based algorithm, which has the best results for this metric. So our model was able to have better performance

than the CF-CBF weighted hybrid, while also reducing its bias toward popular items.
In a cold start scenario, we compared our model with the CBF and the CF-CBF weighted hybrid. Our model largely outperformed the weighted hybrid, and had results similar to those of the CBF algorithm.
As regard the performance on users with different profile lengths, our model achieved larger results improvements than the other algorithms over the users with a long enough profile. This means that it is able to better exploit the amount of information available in the users' profiles. However, the improvements over these users also means that the model produces relatively poorer results over the users with a short profile. A possible future work could be to find the cause, and design a new switching hybrid such that another recommender is used to produce recommendations for these type of users, while our model is employed for the rest.
In conclusion, the approach described in this thesis outperformed the collaborative algorithms, while having performance similar to content based ones in cold start scenario. However, like the other algorithms, it is not able to produce recommendations for new user. Another possible future work can be to integrate demographic information into the graph such that new users can still have three steps paths that lead to item nodes for recommendations.

# Bibliography

[1] P. H. Aditya, I. Budi, and Q. Munajat. “A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for E-commerce in Indonesia: A case study PT X”. In: *2016 International Conference on Advanced Computer Science and Information Systems, ICACSIS 2016*. 2017. ISBN: 9781509046294. DOI: `10.1109/ICACSIS.2016.7872755`.

[2] Charu C. Aggarwal and Charu C. Aggarwal. *Recommender Systems The Textbook*. 2016. ISBN: 9783319296579. DOI: `10.1007/978-3-319-29659-3_3`.

[3] Xavier Amatriain, Josep M. Pujol, and Nuria Oliver. “I like it... i like it not: Evaluating user ratings noise in recommender systems”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009. ISBN: 3642022464. DOI: `10.1007/978-3-642-02247-0_24`.

[4] Fahad Anwaar et al. “HRS-CE: A hybrid framework to integrate content embeddings in recommender systems for cold start items”. In: *Journal of Computational Science* (2018). ISSN: 18777503. DOI: `10.1016/j.jocs.2018.09.008`.

[5] M. Aurnhammer, P. Hanappe, and L. Steels. “Integrating collaborative tagging and emergent semantics for image retrieval”. In: *. . . of the Collaborative Web Tagging . . .* (2006).

[6] Poonam B.Thorat, R. M. Goudar, and Sunita Barve. “Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System”. In: *International Journal of Computer Applications* (2015). DOI: 10.5120/19308-0760.

[7] Stefan Behnel et al. “Cython: The best of both worlds”. In: *Computing in Science and Engineering* (2011). ISSN: 15219615. DOI: 10.1109/MCSE.2010.118.

[8] Léon Bottou. “Large-Scale Machine Learning with Stochastic Gradient Descent”. In: *Proceedings of COMPSTAT'2010*. 2010. DOI: 10.1007/978-3-7908-2604-3_16.

[9] Robin Burke. “Hybrid recommender systems: Survey and experiments”. In: *User Modelling and User-Adapted Interaction* (2002). ISSN: 09241868. DOI: 10.1023/A:1021240730564.

[10] Robin Burke. “Integrating knowledge-based and collaborative-filtering recommender systems”. In: *Proceedings of the Workshop on AI and Electronic Commerce* (1999).

[11] Fabian Christoffel et al. “Blockbusters and wallflowers: Accurate, diverse, and scalable recommendations with random walks”. In: *RecSys 2015 - Proceedings of the 9th ACM Conference on Recommender Systems*. 2015. ISBN: 9781450336925. DOI: 10.1145/2792838.2800180.

[12] Pang Ming Chu and Shie Jue Lee. “A novel recommender system for E-commerce”. In: *Proceedings - 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics, CISP-BMEI 2017*. 2018. ISBN: 9781538619377. DOI: 10.1109/CISP-BMEI.2017.8302310.

[13] Colin Cooper et al. “Random walks in recommender systems: Exact computation and simulations”. In: *WWW 2014 Companion - Proceedings of the 23rd International Conference on World Wide Web*. 2014. ISBN: 9781450327459. DOI: 10.1145/2567948.2579244.

[14] J.S. Cramer. "The Origins of Logistic Regression". In: *SSRN Electronic Journal* (2005). DOI: `10.2139/ssrn.360300`.

[15] Paolo Cremonesi, Roberto Turrin, and Fabio Airoldi. "Hybrid algorithms for recommending new items". In: *Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems, HetRec 2011 - Held at the 5th ACM Conference on Recommender Systems, RecSys 2011*. 2011. ISBN: 9781450310277. DOI: `10.1145/2039320.2039325`.

[16] Marco De Gemmis et al. "Integrating tags in a semantic content-based recommender". In: *RecSys'08: Proceedings of the 2008 ACM Conference on Recommender Systems*. 2008. ISBN: 9781605580937. DOI: `10.1145/1454008.1454036`.

[17] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *COLT 2010 - The 23rd Conference on Learning Theory*. 2010. ISBN: 9780982252925.

[18] Daniel M. Fleder and Kartik Hosanagar. "Recommender systems and their impact on sales diversity". In: *EC'07 - Proceedings of the Eighth Annual Conference on Electronic Commerce*. 2007. ISBN: 159593653X. DOI: `10.1145/1250910.1250939`.

[19] François Fouss et al. "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation". In: *IEEE Transactions on Knowledge and Data Engineering* (2007). ISSN: 10414347. DOI: `10.1109/TKDE.2007.46`.

[20] Simon Funk. "Netflix Update: Try This at Home". In: (2006).

[21] Sergio J. Rojas G., Erik A Christensen, and Francisco J Blanco-Silva. *Learning SciPy for Numerical and Scientific Computing*. 2013. ISBN: 9781782161622. DOI: `10.1017/CBO9781107415324.004`. arXiv: `arXiv:1011.1669v3`.

[22] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. "Beyond accuracy: Evaluating recommender systems by coverage and serendipity". In: *RecSys'10 - Proceedings of the 4th ACM Conference on Recommender Systems*. 2010. ISBN: 9781450304429. DOI: 10.1145/1864708.1864761.

[23] Gebrekirstos G. Gebremeskel and Arjen P. De Vries. "Recommender systems evaluations: Offline, online, time and A/A test". In: *CEUR Workshop Proceedings*. 2016.

[24] Jonathan Gemmell et al. "The impact of ambiguity and redundancy on tag recommendation". In: *RecSys'09 - Proceedings of the 3rd ACM Conference on Recommender Systems*. 2009. ISBN: 9781605584355. DOI: 10.1145/1639714.1639724.

[25] Mustansar Ali Ghazanfar and Adam Prugel-Bennett. "A scalable, accurate hybrid recommender system". In: *3rd International Conference on Knowledge Discovery and Data Mining, WKDD 2010*. 2010. ISBN: 9780769539232. DOI: 10.1109/WKDD.2010.117.

[26] Douglas M. Hawkins. *The Problem of Overfitting*. 2004. DOI: 10.1021/ci0342472.

[27] Jonathan L. Herlocker et al. *Evaluating collaborative filtering recommender systems*. 2004. DOI: 10.1145/963770.963772.

[28] Yifan Hu, Chris Volinsky, and Yehuda Koren. "Collaborative filtering for implicit feedback datasets". In: *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2008. ISBN: 9780769535029. DOI: 10.1109/ICDM.2008.22.

[29] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. "Comparison of implicit and explicit feedback from an online music recommendation service". In: *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems, HetRec 2010, Held at the 4th ACM Conference on*

*Recommender Systems, RecSys 2010*. 2010. ISBN: 9781450304078. DOI: `10.1145/1869446.1869453`.

[30] Christopher Johnson. “Logistic matrix factorization for implicit feedback data”. In: *Advances in Neural Information Processing Systems* (2014).

[31] Diederik P. Kingma and Jimmy Lei Ba. “Adam: A method for stochastic optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. 2015. arXiv: `1412.6980`.

[32] Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix factorization techniques for recommender systems”. In: *Computer* (2009). ISSN: 00189162. DOI: `10.1109/MC.2009.263`.

[33] Semage Laknath. “Recommender Systems with Random Walks: a survey”. In: ().

[34] Tomas Mikolov et al. “Distributed representations ofwords and phrases and their compositionality”. In: *Advances in Neural Information Processing Systems*. 2013. arXiv: `1310.4546`.

[35] Sandeep Nagar. *Introduction to python for engineers and scientists: Open source solutions for numerical computation*. 2017. ISBN: 9781484232040. DOI: `10.1007/978-1-4842-3204-0`.

[36] Tien T. Nguyen et al. “Exploring the filter bubble: The effect of using recommender systems on content diversity”. In: *WWW 2014 - Proceedings of the 23rd International Conference on World Wide Web*. 2014. ISBN: 9781450327442. DOI: `10.1145/2566486.2568012`.

[37] Xia Ning and George Karypis. “SLIM: Sparse LInear Methods for top-N recommender systems”. In: *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2011. ISBN: 9780769544083. DOI: `10.1109/ICDM.2011.134`.

[38] Xia Ning and George Karypis. "Sparse linear methods with side information for top-N recommendations". In: *RecSys'12 - Proceedings of the 6th ACM Conference on Recommender Systems*. 2012. ISBN: 9781450312707. DOI: 10.1145/2365952.2365983.

[39] Patrick Ott. "Incremental Matrix Factorization for Collaborative Filtering". In: *Applied Sciences* (2008).

[40] Elie Parise. "The Filter Bubble: What the Internet Is Hiding from". In: *Policy Perspectives* ().

[41] Arkadiusz Paterek. "Improving regularized singular value decomposition for collaborative filtering". In: *KDD Cup and Workshop* (2007). ISSN: 00121606. DOI: 10.1145/1557019.1557072.

[42] *Recommender systems: An introduction*. 2010. ISBN: 9780511763113. DOI: 10.1017/CBO9780511763113.

[43] Steffen Rendle et al. "BPR: Bayesian personalized ranking from implicit feedback". In: *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI 2009*. 2009. arXiv: 1205.2618.

[44] Francesco Ricci et al. *Recommender Systems Handbook*. 2011. DOI: 10.1007/978-0-387-85820-3.

[45] Dennis M. Ritchie et al. "C PROGRAMMING LANGUAGE." In: *Western Electric engineer* (1981). ISSN: 00433659.

[46] B Sarwar et al. "Application of dimensionality reduction in recommender system-a case study". In: *ACM WebKDD 2000 Web Mining for ECommerce Workshop* (2000). ISSN: 15533514. DOI: 10.3141/1625-22. arXiv: 118.

[47] Markus Schedl et al. "Current challenges and visions in music recommender systems research". In: *International Journal of Multimedia Information Retrieval* (2018). ISSN: 2192662X. DOI: 10.1007/s13735-018-0154-2. arXiv: 1710.03208.

[48] Andrew I. Schein et al. “Methods and metrics for cold-start recommendations”. In: *SIGIR Forum (ACM Special Interest Group on Information Retrieval)*. 2002. DOI: 10.1145/564418.564421.

[49] Giovanni Semeraro et al. “Knowledge infusion into content-based recommender systems”. In: *RecSys'09 - Proceedings of the 3rd ACM Conference on Recommender Systems*. 2009. ISBN: 9781605584355. DOI: 10.1145/1639714.1639773.

[50] Gaurav Tewari, Jim Youll, and Pattie Maes. “Personalized location-based brokering using an agent-based intermediary architecture”. In: *Decision Support Systems*. 2003. DOI: 10.1016/S0167-9236(02)00076-3.

[51] “Using OpenMP: portable shared memory parallel programming”. In: *Choice Reviews Online* (2008). ISSN: 0009-4978. DOI: 10.5860/choice.46-0930.

[52] Stéfan Van Der Walt, S. Chris Colbert, and Gaël Varoquaux. “The NumPy array: A structure for efficient numerical computation”. In: *Computing in Science and Engineering* (2011). ISSN: 15219615. DOI: 10.1109/MCSE.2011.37. arXiv: 1102.1523.

[53] Donghui Wang et al. “A content-based recommender system for computer science publications”. In: *Knowledge-Based Systems* (2018). ISSN: 09507051. DOI: 10.1016/j.knosys.2018.05.001.

[54] Jian Wei et al. “Collaborative filtering and deep learning based recommendation system for cold start items”. In: *Expert Systems with Applications* (2017). ISSN: 09574174. DOI: 10.1016/j.eswa.2016.09.040.

[55] Yeo Chan Yoon and Jun Woo Lee. “Movie Recommendation Using Metadata Based Word2Vec Algorithm”. In: *2018 International Conference on Platform Technology and Service, PlatCon 2018*. 2018. ISBN: 9781538647103. DOI: 10.1109/PlatCon.2018.8472729.