POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# A reproducibility study on Recommendation Systems of papers published in IJCAI and WWW international conferences

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Riccardo Luigi Aielli**

Student ID: 996491
Advisor: Prof. Maurizio Ferrari Dacrema
Academic Year: 2022-2023

# Abstract

Recommender Systems aims at predicting the preferences or scores a user would assign to a set of items by relying on available data. The increasing popularity of these systems is attributed to the growing interest in online platforms aligned with the vast amount of available data. These systems filter the information presenting the user with a precise subset of content ensuring the user's attention is directed towards the most significant items for them.
There is a variety of techniques that can be utilized for generating recommendations. Frequently, these recommendations are crafted through the application of collaborative filtering techniques, which rely on the preference of an extensive user community to predict the most pertinent items for each single customer.
Over the past few years, there has been a notable surge in the exploration of integrating neural techniques such as Deep Learning into the recommender systems domain to enhance the quality of recommendations compared to traditional approaches.
The publication of numerous papers in leading international conferences, asserting superior results surpassing those achieved by traditional methods, has established this category of algorithms as predominant in the research field. Nevertheless, there is no unanimous acknowledgment of this success, and various papers have raised concerns regarding the actual attainment of the claimed results influenced by methodological errors.
In this work, we carried out a comprehensive set of experiments to reproduce the results presented in recently published papers from leading scientific conferences. We assessed the actual performance of newly proposed methods by following a precise and rigorous approach already used in previous studies.
The result of this analysis reveals that none of the reproducible papers consistently outperforms conceptually simple and well-established baseline methods. During our analysis, we identified several methodological issues and some technical errors providing some possible solutions to the reproducibility problem that characterizes the scientific research in this field.

**Keywords:** recommender system, reproducibility, collaborative filtering, neural, deep learning

# Sommario

I sistemi di raccomandazione sono applicativi software che mirano a prevedere le preferenze o i punteggi che un utente assegnerebbe a un insieme di articoli, basandosi sui dati disponibili. La crescente popolarità di questi sistemi è attribuita al crescente interesse di piattaforme online e alla grande quantità di dati disponibili. Questi sistemi filtrano le informazioni, presentando all'utente un sottoinsieme preciso di contenuti e assicurando che l'attenzione dell'utente sia diretta verso gli elementi più significativi per lui.
Esiste una varietà di tecniche che possono essere utilizzate per generare raccomandazioni. Spesso, queste raccomandazioni sono realizzate attraverso l'applicazione di tecniche di collaborative filtering, che si basano sulle preferenze di un'ampia comunità di utenti.
Negli ultimi anni si è assistito a un notevole incremento nell'utilizzo di tecniche neurali come il Deep Learning nel campo dei sistemi di raccomandazione per migliorarne la qualità rispetto ad approcci tradizionali.
La pubblicazione di numerosi articoli nelle principali conferenze internazionali, che affermano risultati superiori a quelli ottenuti con i metodi tradizionali, ha consolidato questa categoria di algoritmi come predominante nel campo della ricerca. Tuttavia, non c'è un riconoscimento unanime di questo successo e diversi articoli hanno sollevato dubbi sull'effettivo raggiungimento dei risultati dichiarati, influenzati da errori metodologici.
In questo lavoro abbiamo condotto una serie completa di esperimenti con l'obiettivo di riprodurre i risultati presentati in articoli pubblicati di recente in importanti conferenze scientifiche. Abbiamo valutato le prestazioni effettive dei nuovi metodi proposti seguendo un approccio preciso e rigoroso già utilizzato in studi precedenti.
Il risultato di questa analisi rivela che nessuno degli algoritmi riprodotti supera in modo consistente i nostri metodi di benchmark concettualmente semplici e consolidati. Durante la nostra analisi, abbiamo identificato diversi problemi metodologici e alcuni errori tecnici fornendo alcune possibili soluzioni al problema della riproducibilità che caratterizza la ricerca scientifica in questo campo.

**Parole chiave:** sistemi di raccomandazione, riproducibilità, collaborative filtering, metodo neurale, deep learning

# Contents

# 1 | Introduction

In recent years recommender systems have become very popular, this is probably due to the growing economic interest in online platforms like Amazon, Netflix, Instagram, Spotify, and others, along with the exponential increase of information available on the web. These systems, driven by algorithms, try to anticipate user preferences or ratings for the available content on these platforms to suggest the right content to the right users. In scenarios such as e-commerce, where suggesting items aligned with buyers' interests, or in online advertising, showcasing content that aligns with user preferences, becomes a means to significantly enhance platform profitability. Simultaneously, it serves to alleviate users from inappropriate information overload, simplifying their path to finding what they are seeking.

In numerous recent research papers on recommender systems, there is a common assertion of substantial advancements compared to the existing "state-of-the-art". Nevertheless, there is no unanimous acknowledgment of this success, and various papers have raised concerns regarding the actual attainment of the claimed results [16].

Some algorithms asserting superior performance compared to the state of the art have undergone inadequate evaluation procedures, resulting in their failure to deliver as claimed. This issue is attributed to the fact that these new algorithms have been evaluated against baseline methods that were either weak or not adequately tuned and optimized. As a support, scientists have demonstrated that less recent methods can outperform the new algorithms if appropriately tuned [54]. Additionally, authors have considerable flexibility in selecting experimental conditions, encompassing metrics, evaluation protocols, datasets, and baseline models. This flexibility often leads authors to present results where their proposed method performs optimally, frequently without offering a motivation for their choices.

In this work, following the same approach used in previous studies [17], we carried out a comprehensive set of experiments with the objective of reproducing the results presented in recently published papers from top-tier scientific conferences. We assessed the actual performance of newly proposed methods by comparing them against appropriately tuned baselines.

The result of this analysis reveals that none of the reproducible papers consistently outperforms conceptually simple and well-established baseline methods. During our analysis, we identified several methodological issues and some technical errors. However, we also observed that the selection of baselines, and the absence of proper optimization for these baselines, are key factors influencing the described results.

## 1.1. Thesis Structure

The structure of the thesis is the following:

- *Introduction*: provides an overview of the work performed describing what are recommender systems and why they are important.
- *State of the art*: short description of the state of the art in the recommender systems field. It describes the categories of algorithms and the evaluation procedures and metrics.
- *Experimental Evaluation Process*: describes the methodology adopted for performing this work and the evaluation framework that has been used in the experiments.
- *Analysis and Results of Reproducible Papers*: provides an extensive description of the papers that have been analyzed and the results obtained to evaluate their performance.
- *Conclusions*: tries to recap what has been done and explain the main challenges we passed through.

# 2 | State of the Art

In this chapter, we are going to describe the state of the art in the domain of Recommender Systems. The goal is to present all the basic concepts and technical aspects of this field in order to give general knowledge about these algorithms. Specifically, we are going to give a more detailed description, needed to clearly understand our work, of those elements that are relevant to the experimental part described in the following chapters.

## 2.1. Recommender Systems

Recommender systems are algorithms that analyze user's behaviors, comprehending their preferences even when not explicitly stated. These systems have gained immense popularity due to the vast amount of information on the Internet. They effectively sift through this information, presenting the user with a precise subset of content ensuring the user's attention is directed towards the most significant content for them, preventing the loss or unnecessary delay in processing information. Furthermore, by prioritizing the right information, recommender systems are able to prevent information overload. This leads to a more efficient utilization of available information. In order to try to predict user preferences recommender systems need data. Within information systems, data quality assumes a critical role, having a significant influence over system performance. When applied to recommender systems, this underscores that recommendation quality hinges not solely on the algorithm in use but also on the quantity and integrity of users and items' data, and their historical interactions. In the presence of flawed, absent, or inaccurate data, the resultant recommendations are bound to be of non-optimal quality [60].
In order to be clear when we mention the word items we are referring to all kinds of content or products the user can interact with. For example, they could be movies, songs, posts, physical products, and so on.
In recommendation system algorithms there are different kinds of data presented in different forms:

- **User Rating Matrix (URM):** This matrix records the ratings users give to the items. This is a matrix representing users as rows and items as columns. This is so

far the most important input for recommender systems because it represents user preferences. It represents the history of the interactions between users and items. There are two different forms of this matrix. It can store explicit ratings which are numbers in a predefined range e.g., from 1 to 5 or from 1 to 10, representing the exact rate the user gives to the item. For example, the review the user gives on TripAdvisor after being in a restaurant. Then there are the implicit ratings which is a binary value of either 0 or 1 indicating whether the user interacted or not with the specific item. For example, if the user watched a movie on Netflix then the value of the cell associated with the user and the movie should be set to one. In general, the explicit ratings represent better the user's taste but they are subject to user bias and difficult to retrieve because the user needs to be involved actively in order to get the ratings, whereas the implicit ones are less informative but are much easier to obtain.
Implicit ratings are becoming preponderant due to their typically higher abundance compared to explicit ratings, they also facilitate real-time adjustments to recommendations, triggered by each click or other forms of user interaction. It is important to notice that in real applications, the number of users and items ranges from hundreds of thousands to millions of users. Each user has usually a few valid ratings and all the other ratings are set to 0 which doesn't mean the user disliked that item but it only means that there is no interaction between the user and the item. This characteristic of a very low density of the matrix is called sparsity. The number of ratings other than zero is frequently lower than 0.01%

$$URM = \begin{array}{c} \\ Francesco \\ Alessia \\ Andrea \\ \vdots \\ Luca \end{array} \begin{array}{c} \begin{array}{ccccc} Tenet & Inception & Interstellar & \cdots & Oppenheimer \end{array} \\ \begin{pmatrix} 2 & 3 & 4 & \cdots & - \\ 1 & - & - & \cdots & 1 \\ - & 2 & 2 & \cdots & 2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 3 & 4 & \cdots & - \end{pmatrix} \end{array}$$

- **Item Content Matrix (ICM):** This matrix represents attributes of items such as movies, songs, books, social network posts, and many others (e.g., for a movie the following are all possible attributes: year, title, genre, director, etc.). In this table, each row represents an item and each column corresponds to one attribute. In the simplest version of the table, the values can be either 0 or 1. It is assigned 1 if the

attribute belongs to the item instead it is 0 if the item doesn't have the attribute. There is also a more complex version of the ICM with values ranging from 0.0 to 1.0 passing through all the decimal values between them in order to specify how important is a specific attribute for the specific item. For example, if an actor is the main character his values should be fixed at 1.0 instead if he has a less important role he could have a value of 0.4.

$$ICM = \begin{array}{c} \\ Tenet \\ Inception \\ Interstellar \\ \vdots \\ Oppenheimer \end{array} \begin{array}{c} \begin{array}{ccccc} Action & Drama & Sciencefiction & \cdots & Musical \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & \cdots & 0 \\ 1 & 0 & 1 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 1 & \cdots & 0 \end{pmatrix} \end{array}$$

- **User content Matrix (UCM):** This matrix represents the characteristics of users. It is similar to the ICM but this time the attributes are referred to the users. Each row is associated to one user and each column is associated to one attribute (e.g., age, nationality, gender etc.)

## 2.2. Recommender Systems Algorithms

We already explained what recommender systems are and what is their input. This section explains several known methods in order to understand their main peculiarities. Recommender systems can be classified according to their main characteristics, their taxonomy is represented in Figure 2.1.

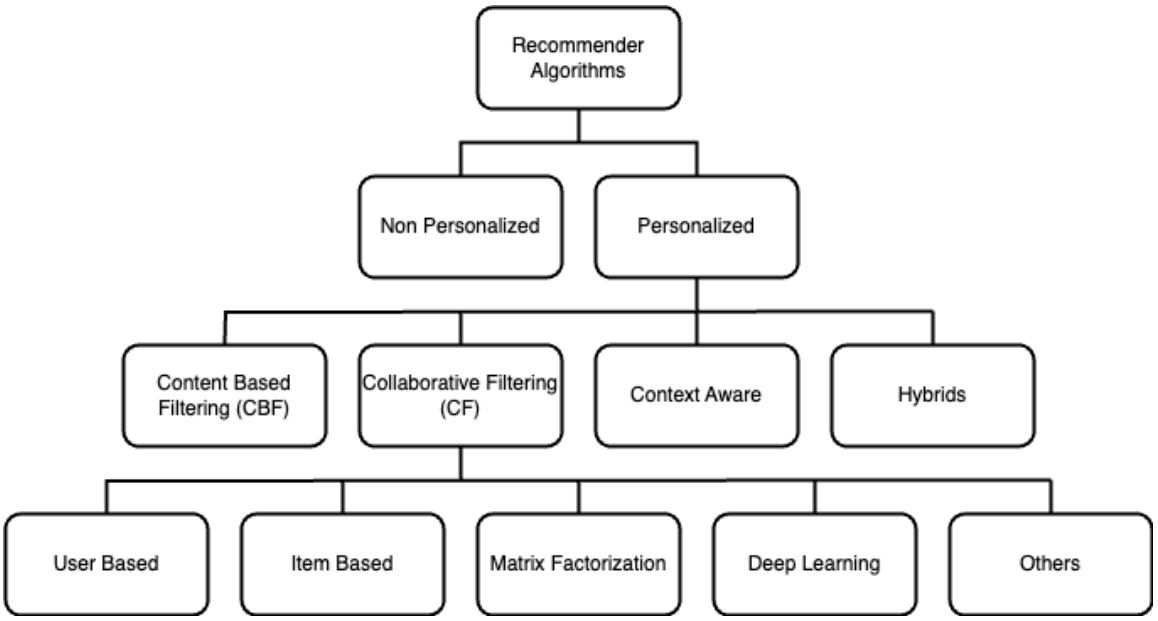


Figure 2.1: Taxonomy of recommender systems' algorithms

### 2.2.1. Non-Personalized

Non-Personalized algorithms provide the same recommendations to all users. This means that different users with different tastes and preferences will get the same items recommended. Usually, a personalized recommended system provides better recommendations but there are some use cases in which non-personalized algorithms are a good choice. For example, they can be used to recommend items to new users or they can be used to display items to non-logged-in users. Due to their simplicity, they are often used as benchmarks for evaluating other more complex algorithms. Three main non-personalized algorithms are:

- **Top-popular:** this algorithm ranks all catalog items based on the frequency of user interactions, and subsequently, it recommends only the top popular items to users. It's evident that this approach relies on user data (URM) to carry out the item ranking.
- **Top-rated:** similar to top-popular but this time items are ranked based on the average ratings provided by users.
- **Random:** a random subset of items is suggested to users from the entire catalog. While it might appear seemingly arbitrary, this approach can serve as a benchmark for assessing other recommendation algorithms.

### 2.2.2. Content-Based Filtering

A second possible distinction among different recommender algorithms other than the level of personalization is about the use of items attributes (ICM). Content-based filtering (CBF) methods [47] [39] [68] leverage the user profile and item characteristics to recommend items. The goal is to recommend items that closely resemble those the user has previously engaged with. This method is based on the assumption that if a user likes item $i$, an item $j$ is similar to $i$, probably that user is going to like also item $j$. The input of this method is the ICM and in some cases also the UCM can be leveraged. The basic idea is to estimate the rating that the user would give to an item by computing the weighted average over the similarity weight of the ratings given to similar items. To generate a measure of how similar two items are there is the similarity function. There are different ways to define a similarity function. In this paragraph, we present the cosine similarity.

$$s_{ij} = \frac{\mathbf{i} \cdot \mathbf{j}}{|\mathbf{i}|_2 \cdot |\mathbf{j}|_2 + C} = \cos\theta \tag{2.1}$$

As shown by the Formula 2.1 item similarities $s_{ij}$ can be calculated by taking the dot product of the vectors $\mathbf{i}$ and $\mathbf{j}$ of the attributes of item $i$ and item $j$ respectively, taken from the rows of the ICM that correspond to the two items. This essentially counts the shared attributes between the two items. Typically, the dot product is normalized to yield the value within the range of 0 to 1, achieved by dividing it by the product of the norms of the two items. In other words, it generates the similarity using the cosine of the angle $\theta$ between the two vectors, this is why this similarity is known as cosine similarity. Additionally, a constant value, denoted as C, referred to as the "shrinking term", is introduced to the product of the norms. This adjustment helps diminish similarity when numerous attributes are unspecified. Without this term, two items with just one attribute with the same value specified would be considered identical. All the similarity scores among pairs of items can be represented with a square matrix $S$ known as the similarity matrix:

$$S = \begin{pmatrix} s_{11} & s_{12} & \dots & s_{1n} \\ s_{21} & s_{22} & \dots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \dots & s_{nn} \end{pmatrix} \tag{2.2}$$

The estimated ratings $\tilde{r_{ui}}$ for user $u$ of item $i$ is computed by the following formula:

$$\tilde{r}_{ui} = \frac{\sum_j r_{uj} \cdot s_{ji}}{\sum_j s_{ji}} \tag{2.3}$$

The main problem of this approach is that there will be only a few items $j$ that are truly similar to the target item $i$, and this will reflect into $S$ with the presence of many values close to 0. All these values are outliers, items that have nothing to do with our target and therefore they contaminate our recommendations. Therefore, we can set all those meaningless values to 0: this is the basic idea behind the KNN (k-Nearest Neighbours) technique.

## K-Nearest Neighbors (KNN) CBF

One of the prevalent approaches in content-based recommender systems is the K-Nearest Neighbors (KNN) algorithm. This method requires the definition of a similarity function to calculate the similarity between every combination of users and items. This technique takes the initial similarity matrix, and, for each row, it keeps only the K most similar items to our target (the nearest neighbours), identified with the K highest similarity values, setting all the others to 0. After this procedure, we will obtain a much sparser matrix that performs better in terms of computation and quality of recommendations. The following formula express in a mathematical notation our reasoning:

$$\tilde{r}_{ui} = \frac{\sum_{j \in kNN(i)} r_{uj} \cdot s_{ji}}{\sum_{j \in kNN(i)} s_{ji}} \tag{2.4}$$

K is a hyperparameter that influences the quality of recommendation and has to be properly tuned. Depending on whether we are considering user or item features, we can distinct two different approaches:

- **UserKNN CBF:** With this approach the similarity is calculated between users. The value is computed by leveraging the UCM matrix using the user attributes as vectors. Since we are using user attributes the similarity obtained is a value indicating how two different users have the same taste. Since we have a user-user similarity, we can compute the estimated rating vector $\tilde{r}_u$ of user u as:

$$\tilde{\mathbf{r}}_\mathbf{u} = \mathbf{s}_\mathbf{u} \cdot URM \tag{2.5}$$

  where $s_u$ is the similarity vector of user $u$ with respect to all the other users keeping only the top knn interaction.

- **ItemKNN CBF:** With this approach the similarity matrix $S$ between all item pairs is computed by leveraging the ICM matrix using the item features as vectors. Since we are using item features the similarity obtained is a value indicating how two different items are similar. Since we have an item-item similarity, we can compute the estimated rating vector $\tilde{r}_u$ of user u as:

$$\tilde{\mathbf{r}}_\mathbf{u} = \mathbf{r}_\mathbf{u} \cdot S^T \tag{2.6}$$

  where $r_u$ is the rating vector of user $u$ and $S^T$ is the similarity matrix in wich only the top knn interactions are kept.

Item content based recommenders, in contrast to collaborative filtering approaches, are not affected by the cold-start problem for new items. In fact they only relay to features wich are know since the beginnig. On the other hand they provide a limited degree of novelty due to the matching between user profile and item features.

### Non-Binary Feauture Weight

As we have seen each item is represented by a feature vector that corresponds to various attributes of the item, such as genres, actors, directors, and more, in the case of the movie domain. These features wich are in the ICM can take binary or decimal values. While calculating similarity simple measures of similarity like the dot product or cosine similarity described before assume equal importance for all features. However, in practice, when users assess the similarity between two items, they often assign varying degrees of significance to different attributes. For instance, when selecting a movie, genre attribute may carry more weight than the producer attribute. It can be stated that users base their judgments on same latent criteria, which represent a weighted linear combination of attribute differences. The objective of feature weighting is to estimate these weights to enhance the performance of a recommendation model [18]. The easiest approach to this problem is to give different weights with respect to different items features in such a way to highlight the most important ones and let them contribute more to the computation of the similarity. This weights can be considered as hyperparameters and as so they need to be properly tuned during the validation phase of the model. Other methods are employed for more advanced feature weighting, with primary inspiration drawn from the field of information retrieval. These approaches often incorporate weight functions that are related to the notion of term frequency. One of these approaches is Term Frequency - Inverse Document Frequency (TF-IDF) [12]. This approach come from the text mining domain. The general idea of this concept is that in a collection of documents a term

is more relevant if the frequency of a specific term is high in a document and at the same time appears with low frequency in the other documents. The same concept can be applied in the feauture weight problem of the reccomander systems domain to balance the weights of attributes, depending on their frequency of appearance in the items. The TF-IDF value of an attribute is given by the product of two distinct weights:

- **Term Frequency:** The Term Frequency (TF) represents how much important is an attribute $a$ in the description of an item $i$ and it is calculated as:

$$TF_{a,i} = \frac{N_{a,i}}{N_i} \tag{2.7}$$

  where $N_{a,i}$ is the number of appearances of attribute $a$ in item $i$ and $N_i$ is the number of attributes of item $i$. Since most attributes appear only one time in the description of an item, the values of TF will be small.

- **Inverse Document Frequency:** The Inverse Document Frequency (IDF), on the other hand, represents how much important is an attribute for the whole dataset. This sentence means that IDF has a higher value for rare attributes (attributes that appear few times in the whole dataset) because having that attribute means excluding many items which do not have it. We say that attributes with a high value of IDF have higher informative content, while attributes with a low value of IDF have nearly no informative content because they do not help in the distinction of items. If every item has attribute $a$, then we cannot use it to compare items because they would result identical. IDF is calculated as follows:

$$IDF_a = \log_2 \frac{N_{ITEMS}}{N_a} \tag{2.8}$$

  where $N_{ITEMS}$ is the number of items in the dataset and $N_a$ is the number of items containing attribute $a$. As we can imagine, IDF will usually have high values and thus it counterbalances the low values of TF.

### 2.2.3. Collaborative Filtering

Collaborative Filtering (CF) methods are based solely on the ratings or opinions of a community of users and do not necessitate any item attributes or detailed user profiles, which usually are not easy to obtain due to privacy permission. The accuracy of predictions heavily relies on the quantity of available ratings and for this reason, this method cannot be applied to recommend new products lacking of user ratings. These methods

rely only on the user's explicit and implicit ratings [58]. Collaborative filtering is one the most popular approach to recommended systems for this reason numerous algorithms that make use of CF techniques have been developed [2] [55].

### K-Nearest Neighbors (KNN) CF

The first collaborative filtering technique we will explain is K-Nearest Neighbors CF. The basic idea is similar to the one of KNN CBF in fact in User-Based KNN CF we want to look for users with similar tastes and recommend them the K items they like the most. While in Item-Based KNN CF, we want to look for similar items and recommend the K more similar items to the users who liked that item. The main difference is that now we rely only on User-Items ratings (URM) and not on user or item features (ICM, UCM). Both algorithms are based on a fundamental assumption, which is that if two users exhibit similar ratings for certain items, they are likely to exhibit similar ratings for other items as well. Conversely, if two items receive similar ratings from a subset of users, they are expected to get similar ratings from the other users. These definitions are heavily grounded in the concept of a similarity measure applied to a pair of vectors, which in our context can represent user or item profiles. It's worth emphasizing that both the user-based and item-based approaches are very similar, with the user-item interaction data (URM) being the sole basis for calculating estimated ratings. On one hand, these methods prove to be highly effective, drawing strength from the history of prior user interactions and consistently delivering high-quality recommendations. However, on the other side, they suffer from the cold start problem [36], which arises when there is insufficient information available to make recommendations for a new user or to suggest a new item to any user, especially in cases where many users or items have limited interaction history in their profiles.

### Matrix Factorization

In all the techniques seen so far, we have always considered a model based on similarities between items and items or users and users. Now we want to extract from the URM something acting as a counterpart of a "similarity" between users and items. The approach we are going to explain is known as Matrix Factorization. First of all, we need to introduce something that could connect the preferences of a user to the items: latent factors. Latent factors are only an abstraction made to connect users and items. In fact items are described using latent factors (also called features, but with different meanings to what concerns attributes) with a high value, meaning a higher presence of that feature in describing them. While users express their preferences through features with a high

value meaning a higher preference: therefore, users will like the most, items that contain the features they like the most. The Matrix Factorization (MF) algorithm serves as the foundation for numerous successful implementations [30] of latent factor models, primarily because of its excellent scalability and precision.

From a mathematical point of view the URM with cardinality $|U|\times|I|$, with $|I|$ indicate the number of item and $|U|$ indicate the number of users, is expressed as $AB$ with respective cardinalities $|U| \times |K|$ and $|K| \times |I|$ with $|K|$ representing the number of latent features. $|K|$ should be much smaller than the number of users $|U|$ and the number of items $|I|$.

$$URM \approx AB \tag{2.9}$$

Every row of matrix $A$ defines a representation of a user and every column of $B$ defines a new representation of an item. The primary difficulty lies in calculating the transformation of each item and user into latent vectors. The problem of finding the right number of latent factors $|K|$ and the values of latent factors itself can be seen as a machine learning problem. When we talk about machine learning, the first concept coming to our mind is the loss function, which is a way to compare the real, observed results with the predicted ones. This comparison between real values and predicted ones is closely related to the concept of the quality of a recommender system. It is clear that there is a parallelism between searching for the highest quality and, simultaneously, the lowest value of the loss function. Therefore, by looking from another perspective, we can consider this comparison between estimated ratings and real ratings as a loss function to be minimized, whose goal is to find the best similarity matrix (the model) associated with the lowest possible error.

### Bayesian Probabilistic Ranking

One of the quality metrics that we can optimize to build the model for our recommender system is the Bayesian Probabilistic Ranking (BPR) [53]. This approach is one of the pairwise ranking metrics. It is based on the concept of posterior probability. The idea is that we want to predict the rating of the relevant item in such a way that its estimated rating is greater than the estimated rating for the non-relevant item. A common function to estimate the probability is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

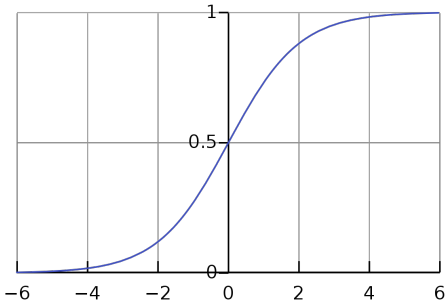


Figure 2.2: Sigmoid function

We know that a high value of $x$ corresponds to a value of the probability function $\sigma(x)$ that is close to 1, so the complete formulation of the probability function is:

$$x_{uij} = \tilde{r}_{ui} - \tilde{r}_{uj} \tag{2.11}$$

$$\sigma(x_{uij}) = \frac{1}{1 + e^{-x_{uij}}} \tag{2.12}$$

Where $\tilde{r}_{ui}$ is the predicted rating of relevant item $i$ for user $u$, $\tilde{r}_{uj}$ is the predicted rating of non-relevant item $j$ for user $u$ and $x_{uij}$ is the pairwise difference between item $i$ and item $j$

## PureSVD

This method is linked to Matrix Factorization (MF) technique, it involves the factorization of the User-Item Rating Matrix (URM). SVD stands for Singular Value Decomposition. This method is based on identifying latent semantic factors in information retrieval. PureSVD is able to outperform more complex latent factor models as shown in [13] therefore is a very good algorithm. Having $|U|$ as the number of users, $|I|$ as the number of items and $|F|$ as the number of latent factors, it computes the estimated user rating matrix $U\tilde{R}M$ by the following factorization:

$$U\tilde{R}M = G \cdot \Sigma \cdot Q^T \tag{2.13}$$

where $G(|U| \times |F|)$ and $Q(|I| \times |F|)$ are two orthonormal matrices and $\Sigma(|F| \times |F|)$ is a diagonal matrix with the first $F$ singular values. Being G and Q orthonormal, the algorithm can be expressed by the following formula:

$$P = G \cdot \Sigma = URM \cdot Q \tag{2.14}$$

The recommendation is generated by assessing a specific vector within the previously calculated matrix. Consequently, the estimated user rating vector for the user $u$ is computed as follows:

$$\tilde{\mathbf{r}}_\mathbf{u} = \mathbf{g}_\mathbf{u} \cdot \Sigma \cdot Q^T \tag{2.15}$$

The primary limitation of SVD models is the prevalence of significant sparsity in the majority of datasets, particularly in real-world scenarios. In such conditions, SVD-based models struggle to gain meaningful insights and produce results that are almost useless.

## Graph Based Algorithms

These techniques are based on the concept of "graph", as the name explains very well. We can represent users and items as nodes and interactions as links between them. To represent this concept we need to adopt an undirected bipartite graph because we know that users and items are two different groups endowed with different characteristics and that relationships can happen only between a user and an item. We can use either implicit or explicit ratings. In the case of explicit ratings, interactions can be distinguished between negative and positive. In this case, we have weights on the interaction edges of the graph linking user and item nodes, which are the ratings given by users. Figure 2.3 exemplifies this concept.

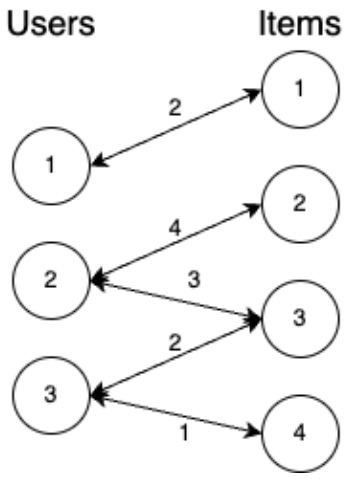


Figure 2.3: Bipartite graph with explicit ratings.

In the case of the bipartite graph with implicit ratings, weights are set to 1. For the mathematical representation of the model, we introduce an adjacency matrix denoted as $A$. This matrix conceptually serves as a representation of the bipartite graph, with each row corresponding to a user node and each column representing an item node. The definition is as follows:

$$A_{u,i} = \begin{cases} w_{u,i}, & \text{if the edge from } u \text{ to } i \text{ exists} \\ 0, & \text{otherwise} \end{cases} \tag{2.16}$$

Where $w_{u,i}$ indicates the weight of the edge from user $u$ to item $i$ and 0 means there is no interaction between those two nodes. Graph-based methods offer high-quality recommendations with low computational costs. This is primarily attributed to their capability to query and traverse the relationships modeled by the graph structure. In the process of item recommendation for a particular user, a scoring algorithm arranges the items based on the similarities derived from the connections between the user's vertex and the item vertices, as defined by the graph's structure. The outcoming idea is to follow the links in the graph in order to find possibly correlated items or items that users could like. A widely used approach for computing similarities involves conducting "random walks" within the graph, which entails random transitions along the edges between nodes, starting from the user's representative node. To depict these transitions, a transition matrix is employed. Each entry [i,u] represents the probability of moving from node i to node u in a single step. The precise distribution of random walks following a specified number of steps can be determined through algebraic matrix operations or approximated by simulating multiple random walks. Consequently, the similarity between two items is assessed based on the transition probability of a random walk commencing from one item and arriving at the other. Because the ranking algorithm is based on the ratings between users and items rather than user and item profiles, this category of methods is classified within the collaborative filtering (CF) class.

### P3alpha

*P3alpha* is an extension of the P3 algorithm, which assesses the ranking of items for a user, based on the distribution of the third vertex in a random walk that originates from node $u$ [11]. In this context, an item $i$ is ranked higher than an item $j$ if there's a greater probability that the random walk, at its third step, arrives at node $i$ rather than node $j$. To estimate the probability of passing from user $u$ to item $i$ we use the transition matrix denoted as $P$, it can be obtained by normalizing row by row the adjacency matrix $A$ with

the following formula:

$$P_{u,i} = \frac{A_{u,i}}{\sum_{k \in I} A_{u,k}} \tag{2.17}$$

*P3alpha* implements a tree-step random walk, transitioning from users to items, then from items to users, and lastly from user to item again. We can represent the probability distribution of reaching an item $i$ starting from a user $u$ at the third step as follows:

$$U\tilde{R}M = P_{u,i} \cdot P_{i,u} \cdot P_{u,i} \tag{2.18}$$

where $P_{u,i}$ is the user-item transition matrix and $P_{i,u}$ is the item-user transition matrix. If we normalize the $URM$ row by row we obtain the $P_{u,i}$, similarly if we normalize the $URM^T$ row by row we obtain $P_{i,u}$. While computing the probabilities of transitioning between users and items *P3alpha* raise the normalized ratings to the power of $\alpha$ which is an hyperparameter used to control the normalization of the probability values. The complete formulation is:

$$\tilde{\mathbf{r}}_{\mathbf{u}} = \mathbf{r}_{\mathbf{u}} \cdot \left(URM^T \cdot URM\right)^{\alpha} \tag{2.19}$$

where $\mathbf{r_u}$ is is the row of the normalized $URM$ corresponding to user $u$. Typically, as the product of the URM and its transpose results in a dense matrix, a top-K nearest neighbor selection is employed on the columns of the resulting matrix, a process analogous to what is done in K-nearest neighbors (KNN) algorithms. The hyperparameter $\alpha$, in conjunction with the neighborhood size K, plays a crucial role in the algorithm. Moreover, there are no relevant gains in the quality of recommendations if we perform more than three jumps.

### RP3beta

The main risk while adopting *P3alpha* as denoted by [46] is that it tends to recommend popular items because there is a higher probability of jumping to nodes with many links rather than to nodes with only a few links. *RP3beta*, tries to fix this "popularity bias" by penalizing the "probability" values of popular items with a coefficient $\beta$, which is a positive hyperparameter, reducing the reachability gap between popular and unknown items. Within *RP3beta*, the similarity between two items is determined through the use of *P3alpha*

$$\tilde{\mathbf{r}'}_{\mathbf{u}} = \frac{\tilde{\mathbf{r}}_{\mathbf{u}}}{\mathbf{d}^{\beta}} \tag{2.20}$$

Where $\tilde{\mathbf{r}}_{\mathbf{u}}$ is the estimated ratings vector of user $u$ form $P3alpha$, $\mathbf{d}$ is the in-degree of item $i$ i.e., the number of incoming edges to node $i$ in the bipartite graph which represents the item's popularity. Notably, when $\beta$ is set to 0, $RP3beta$ becomes equivalent to $P3alpha$.

### GraphFilter CF

GraphFilter CF is a simple yet effective and computationally efficient graph-based baseline algorithm for collaborative filtering that has been introduced in [61]. Given an implicit feedback matrix, GraphFilter CF can be obtained in a closed form instead of expensive training with back-propagation, as a result, its training is particularly efficient. It is based on ideal low-pass filters to guarantee a high computational efficiency. As shown in [61] the ideal low-pass filter is equivalent to an infinite layer graph convolutional network (GCN) without layer combination. It suffers from over-smoothing, which means that it lacks low-order information in the graph. As a result, combining the linear filter and ideal low-pass filter will result in a strong baseline. Denoting the all one column vector of any dimension as $\mathbf{1}$, and degree matrix $D_I$ as:

$$D_I = Diag\left(1^T R\right) \tag{2.21}$$

where $R$ is the rating matrix we can formulate this algorithm with the following formula:

$$\tilde{\mathbf{r}}_{\mathbf{u}} = \mathbf{r}_{\mathbf{u}}\left(\widehat{R}^T\widehat{R} + \alpha D_I^{-\frac{1}{2}}\bar{U}\bar{U}^T D_I^{\frac{1}{2}}\right) \tag{2.22}$$

where $\tilde{r}_u$ denote the predicted scores and $\mathbf{r}_{\mathbf{u}}$ denote the observed scores. $\widehat{R}$ is the normalized rating matrix, $\bar{U}$ is the top-K singular vectors of $\widehat{R}$ and $\alpha$ is a tuned hyperparameter.

## 2.2.4. Hybrid

Hybrid recommender systems try to merge both collaborative filtering (CF) and content-based filtering (CBF) techniques to leverage the complementary advantages of both approaches. Numerous studies [6] that have examined the performance of hybrid systems in contrast to pure collaborative and content-based methods have shown that hybrid recommenders tend to deliver more precise and varied recommendations than their purely CF or CBF counterparts. Moreover, these hybrid methods can effectively address common

challenges in recommender systems, including the cold start problem. We can distinguish different categories according to the strategy used to combine different algorithms:

- **Input Data Concatenation:** The input data from collaborative filtering and content-based filtering are concatenated in order to generate a new input matrix. Then the input data is fed to the recommended algorithm. In some cases, the algorithm needs to be adapted to work with this new input data.
- **Merging Models:** To create a hybrid merging model ensemble the two algorithm needs to share the same model structure. The idea behind it is to merge the partial result of the model. For example, if we want to merge the ItemKNN CF and the ItemKNN CBF we can merge the two similarity matrices in order to generate a new one with the embedded knowledge of both algorithms, increasing the performance of the models.
- **Merging Estimated Ratings:** In this last category, the idea is to merge the estimated ratings of different models by combining the outputs with different weights and techniques.

## 2.3. Deep Learning

Deep learning is a branch of machine learning which is based on Artificial Neural Networks (ANN). It is capable of learning complex patterns and relationships within data. It has become increasingly popular in recent years due to the advances in processing power and the availability of large datasets although the first references are from the 1960s

### 2.3.1. Artificial Neural Networks

Artificial Neural Networks (ANN) are inspired by the structure and function of the human brain's biological neurons, and they are designed to learn from large amounts of data. ANNs are computational systems composed of interconnected units referred to as neurons. These neurons are based on a well-established mathematical model known as the Perceptron, originally studied and developed by Rosenblatt in 1958 [32].

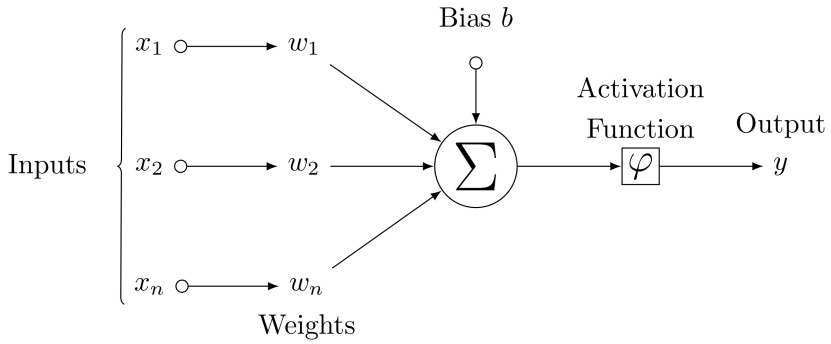


Figure 2.4: Schematic representation of the Perceptron

In this network, input signals pass through a weighting process, then they are aggregated, and finally, they pass through an activation function. If the accumulated signal surpasses a specific threshold, the neuron is said to activate:

$$y(x) = \varphi\left(b + \sum_{i=0}^{n} w_i x_i\right) = \varphi\left(b + w^T x\right) \tag{2.23}$$

Where $b$ is the bias and $\varphi$ is the activation function. Common activation functions are: Step Function, Sign Function, Sigmoid Function, Rectified Linear Unit (ReLU), and Hyperbolic Tangent Function.

An Artificial Neural Network is structured using organized sets of Perceptrons, referred to as layers. During the learning phase, the network's neurons' weights and biases are fine-tuned to minimize the disparity between the network's output and the sample data. This adjustment is typically facilitated by a gradient-based optimization technique known as back-propagation. Inputs are directed into an input layer, consisting of a number of neurons corresponding to the input dimensions, while the outcomes are produced from an output layer with a number of neuron matching the number of desired outputs.

The process of learning can be categorized into three main types:

- **Supervised Learning:** Which involves mapping input to output using the provided example input-output pairs. This method to work well needs high-quality labeled and annotated data.
- **Semi-Supervised Learning:** Which utilizes a limited amount of labeled data alongside a substantial amount of unlabeled data during training.
- **Unsupervised Learning:** Which operates without pre-existing labels and with minimal human intervention. The model attempts to find patterns and relationships

in the data without being given explicit feedback in the form of labels.

### Self-Supervised Learning (SSL)

In the last category belongs Self-Supervised Learning which is an algorithm paradigm used to train artificial intelligence based models. It is used when models are provided vast amounts of raw and completely unlabeled data trying to generate the labels themselves. Due to the need for non-labeled data, it is particularly scalable when larger datasets are used. One example of this paradigm is Contrastive Learning SSL.
Contrastive Learning is a Machine Learning paradigm where unlabeled data inputs are juxtaposed against each other to teach the model which inputs are similar and which are different. As the name suggests, samples are contrasted against each other, and those belonging to the same distribution are pushed toward each other in the embedding space representation. In contrast, those belonging to different distributions are pulled against each other. The efficacy of SSL methods as compared to traditional supervised learning methods has captured the attention of several machine learning researchers.

## 2.3.2. Deep Neural Networks

With the term Deep Neural Networks (DNN) [42] we refer to Artificial Neural Networks that have multiple layers between the input and output ones. These layers are typically called hidden layers.

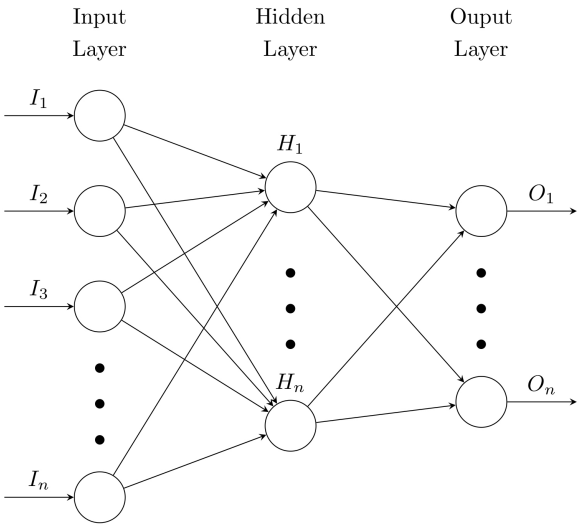


Figure 2.5: Schematic representation of an example of DNN, in which one hidden layer is added between input and output layers.

- **Feedforward Deep Neural Networks (FNN):** The feedforward neural network is one of the simplest neural networks. Its flow is uni-directional, meaning that the

information in the model flows in only one direction. From the input nodes, through the hidden nodes, and to the output nodes. There aren't any cycles or loops. Modern feedforward networks are trained using the backpropagation method.

- **Recurrent Deep Neural Networks (RNN):** In contrast to Feedforward Neural Networks, Recurrent Deep Neural Networks have a state and a recursive flow. This enable the network to function with a sort of memory, examples of which include Long-Short Term Memory (LSTM) and Recurrent Gated Unit (GRU) cells.
- **Convolutional Deep Neural Networks (CNN):** They employ convolution operations instead of the more general matrix multiplication in at least one of their layers, typically composed of an input layer, an output layer, and multiple hidden layers. These models find significant utility in computer vision tasks like in image and video recognition.

### 2.3.3. Deep Learning And Recommender Systems

In recent times, Deep Learning has gained significant attention across various research domains, including computer vision and natural language processing. Its remarkable performance and capability to learn feature representations from scratch have fueled its popularity. Moreover, Deep Learning has made a notable impact on the field of recommender systems [9], particularly in addressing data sparsity challenges in traditional collaborative filtering algorithms, while enhancing the overall performance [24]. It is a very powerful instrument even if it is not omnipotent in fact there are scenarios where it does not always work well [16].
One application of deep learning to reccomander system is through the use of autoencoders. Autoencoders serve the purpose of acquiring lower-dimensional feature representations and making predictions for the absent ratings within the User Rating Matrix. These autoencoders can be broken down into two essential components: the encoder, which condenses the data through a bottleneck, and the decoder, which converts the encoding back into its original shape while minimizing errors as much as possible. Since a dimension reduction is involved, the neural network must acquire a representation in a lower dimension of the input, known as the latent space, to effectively reconstruct the input. Autoencoders can be used to acquire feature representations from user-item interactions, which can subsequently be employed in constructing collaborative filtering (CF) models.
Another interesting application of deep learning in recommender system is to what concerns embeddings. These approaches aim to develop models that can encapsulate users,

items, or contextual profiles within latent spaces. These embeddings can be employed directly for offering recommendations or as input for other Deep Learning techniques that generate recommendations. This concept finds application in Multi-View Deep Neural Networks and Neural Collaborative Filtering [24]. Convolutional Neural Networks are often utilized to extract features from high-dimensional data.
Another method considers the notion of sequences or sessions which is the interaction of users in a limited time-space interval. It is in this context that Sequence-Aware models are useful. Given that some e-commerce platforms and a lot of news and media websites typically do not force the user to log in, the objective is to discern browsing patterns within a single-user session. The prevailing strategy involves modeling sessions as sequences of events, corresponding to the user's interactions with the platform. For such tasks, Recurrent Neural Networks have emerged as one of the most effective architectures, demonstrating remarkable performance in modeling sequences of data.

## 2.4. Evaluation of Recommender Systems

The evaluation of a RS consists in the analysis of the information collected during a series of experiments. The analysis is performed applying a series of metrics that allow to evaluate the single aspects that characterize the performance of the system.

### 2.4.1. Online Evaluation

With online evaluation, feedbacks are provided by users who tested the recommender system and the feedbacks are used to check if its quality indicators expectations are met. Some of the methods used are:

- Direct User Feedback: Consists of asking the user for feedback, for example through questionnaires. The quality of the feedback may be poor since usually questionnaires bother users. Moreover, the size of the sample should be statistically relevant.
- A/B Testing: It is used to compare two different recommender systems. The system is assessed within its actual operational environment, and real user feedback is collected by their behaviour through the system.
- Controlled Experiments: The engagement is restricted to an aware small group of users involved in a controlled environment. At the end of the experiments they will be asked to provide feedbacks.

### 2.4.2. Offline Evaluation

Unlike online evaluation, there is no direct engagement with real users in this approach. The assessment is conducted by giving as input to the system's pre-existing data of user-item interactions. A considerable majority of the algorithms presented in scientific papers and conference publications are evaluated using this methodology. The idea is to simulate the situation in which the system makes recommendations and evaluates them.

#### Evaluation Metrics

In the realm of recommender systems, evaluating algorithm performance presents a complex challenge. Various issues must be considered when selecting the appropriate evaluation metrics to evaluate a specific algorithm's effectiveness. Firstly, it's crucial to recognize that each algorithm might be tailored to address a highly specific task. Consequently, it's important to employ the right metrics for accurate evaluation. Another significant factor to consider is the existence of a multitude of datasets within the field, each characterized by its unique properties. Over the past years, a multitude of metrics has been published and employed for the quantitative assessment of recommender system accuracy [26]. Consequently, a lack of standardization and a huge number of metrics now exist in the field.

#### Prediction Metrics

Prediction metrics are employed to assess the degree of proximity between the predicted rating and the actual rating, also known as the ground truth.

- **Mean Absolute Error (MAE):** The Mean Absolute Error is computed as the average of absolute errors, where absolute errors represent the absolute differences between the predicted values and the actual values.

$$MAE = \frac{1}{U}\sum_{u=1}^{U}\sum_{i=1}^{|r_u|}|r_{ui} - \tilde{r}_{ui}| \tag{2.24}$$

  Where $U$ is the number of users in the test set, $|r_u|$ is the cardinality of relevant items (non-zero ratings i.e., the ground-truth) for user $u$ and $r_{ui}$ and $\tilde{r}_{ui}$ are respectively the real value and the predicted value.

- **Mean Squared Error(MSE):** This measure is close to the MAE and it is more popular because it sums up the square of the absolute value. It is easier to write

an algorithm that minimizes a squared function than one that minimizes a function with an absolute value.

$$MAE = \frac{1}{U}\sum_{u=1}^{U}\sum_{i=1}^{|r_u|}(r_{ui} - \tilde{r}_{ui})^2 \tag{2.25}$$

Two other measures related to MAE and MSE are the Root Mean Squared Error(RMSE) and Normalized Mean Absolute Error(NMAE). All these approaches make the "Missing As Random" (MAR) assumption, which means that the distribution of unknown ratings is considered to be identical to the distribution of ratings in the ground truth.

## Classification Metrics

Classification Metrics do not rely on differences between estimated and real ratings but on the concept of relevant items. Classification metrics rely on the concept of accuracy and are therefore well-suited for identifying high-quality items to recommend. In recommender systems, it's common practice to assess a particular metric with a specified cutoff value denoted as "k" (@k). This implies that the evaluation focuses solely on the top-k recommended items, rather than considering predictions for the entire set of items.

- **Precision @k:** It is defined as the ratio of correct recommendations to the total number of recommendations provided:

$$Precision@k = \frac{relevant\ items \cap recommended\ items}{recommended\ items} \tag{2.26}$$

- **Recall @k:** It is defined as the number of relevant items recommended divided by the number of relevant items. It is the percentage of positive instances correctly detected by the system.

$$Recall@k = \frac{relevant\ items \cap recommended\ items}{relevant\ items} \tag{2.27}$$

- **F-measure @k:** This measure is also called F1 and is defined as the harmonic mean of precision and recall:

$$F\text{-}measure@k = 2 \cdot \frac{Precision@k \cdot Recall@k}{Precision@k + Recall@k} \tag{2.28}$$

- **Hit Rate @k:** The Hit Rate (HR) is the total number of relevant items recom-

mended (hits) divided by the number of users. The number of hits is the number of items in the test set that were also present in the top-k recommendation for each user:

$$HitRate@k = \frac{relevant\ items \cap recommended\ items}{number\ of\ users} \tag{2.29}$$

With these metrics, we are comparing labels instead of values which means that unknown ratings are labeled as "non-relevant". This means that we are making the "Missing As Negative" (MAN) assumption, which is more conservative than the MAR one.

## Ranking Metrics

Ranking metrics assess the capacity of a recommendation algorithm to generate a sorted list of items, mirroring the user's preferences for the same items. In other words, they try to measure how much a user likes an item compared to other items [26]. In this context, the precisely estimated rating for an item holds less significance. What matters is the presence of relevant items in the list and, their relative positions. Unlike classification metrics, these metrics come into play when the objective is to create a list of items in which the relative ordering significantly affects the quality of the recommendations.

- **Mean Average Precision @k (MAP@k):** This metric represents the mean, computed over all users, of the averages of the precisions computed at each cutoff (from 1 up to k), which are the average precision scores:

$$MAP@k = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{\sum_{i=1}^{k} Precision@i\left(\tilde{R}_u\right) \cdot I\left(\tilde{R}_{ui} \in R_u\right)}{|R_u|} \tag{2.30}$$

  Where $R_u$ is the set of relevant items and $\tilde{R}_u$ and $\tilde{R}_{ui}$ are the sets of recommended items. $I(x)$ is the Indicator function ($I(x) = 1$ if $x = true$, else 0)

- **Mean Reciprocal Rank @k (MRR@k):** is the average value of the scores made in the following way. For each recommendation made by the recommender system, the position of the first correct item, assigning a score equal to the inverse of its position in the list e.g., 1 if the first correct item is in the first place, $\frac{1}{2}$ if the first correct item is in the second place, and so on.

$$MRR@k = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{rank_i} \tag{2.31}$$

$rank_i$ indicates the position in the ranking of the first relevant item for user u.

- **Average Reciprocal Hit Rank @k (ARHR@k):** This metric is widely used to assess how close the first relevant item is to the beginning of the recommendation list. It assigns varying degrees of importance to each hit based on its position in the ranked list. In the calculation of the average score, hits appearing earlier in the list carry more weight. The resulting score typically falls within the range of 0 to 1, with higher values indicating that the algorithm has a better ability to predict items that appear at the top of the list.

$$ARHR@k = \frac{1}{n}\sum_{i=1}^{h}\frac{1}{rank\,(i)} \tag{2.32}$$

where $n$ is the number of relevant items, $h$ is the number of hits and $rank\,(i)$ is the position in the ranked list of the item for the $i$-th hit.

- **Normalized Discounted Cumulative Gain @k (NDCG@k):**

This metric evaluates the quality of ranking by considering the placement of accurate recommendations within the list. It assigns a score to each item that diminishes logarithmically as the item's position in the prediction list moves further down. To obtain NDCG, we need to normalize the value of Discounted Cumulative Gain (DCG) by dividing it by the ideal DCG (IDCG).

$$NDCG@k = \frac{DCG@k}{IDCG} \tag{2.33}$$

$$DCG@k = \sum_{i=1}^{k}\frac{2^{rel_i}-1}{\log_2(i+1)} \tag{2.34}$$

where $rel_i$ is the graded relevance, for the item ranked at position $i$ ("1" for implicit ratings and the rating itself in case of explicit ratings).

IDCG is determined using the identical expression employed for calculating DCG for a specific user. However, in contrast to evaluating all items in the recommendation list, each item is assigned its true value, which is derived from the test data.

$$IDCG = \sum_{i=1}^{n}\frac{2^{rel_i}-1}{\log_2(i+1)} \tag{2.35}$$

## Beyond Accuracy Metrics

While classification and ranking metrics assess the system's capacity to offer all relevant recommendations in the right sequence, beyond-accuracy metrics are employed to study its capability to offer diversified recommendations to various users [17] [1]. Diversity metrics can be categorized into two distinct groups: individual diversity and aggregate diversity. The former evaluates from a user's perspective and is computed for each distinct recommendation list, while the latter regards the system as a whole and is determined by considering the recommendations delivered to all users.

- **Novelty**

  This metric introduced by [69] is designed to quantify the novelty of recommendations. This metric highlights the algorithms that can suggest items from the less common section of the catalog, which users are less likely to be familiar with. It is formulated as follows:

$$Novelty = -\frac{1}{|S_{test}| \times k} \cdot \sum_{s \in S_{test}} \sum_{i=1}^{k} \log_2 \left[ freq\left(i_i\right) \right] \tag{2.36}$$

  Where the function "freq" returns the normalized frequency of a certain item, which is the ratio of the number of ratings associated with the item divided by the overall number of available ratings in the sequence $s \in S_{test}$. We can also set $\log_2 [0] = 0$ to avoid item that has never been rated by any user considered as novel.

- **Item Coverage (Item Cov)**

  This is an aggregate diversity metric that counts the number of items which has been recommended from the system at least once.

$$Coverage = \frac{1}{|I|} \cdot \sum_{i \in I} \text{rec}(i) > 0 \tag{2.37}$$

  Where $|I|$ is the total number of items and the function $\text{rec}(i) > 0$ returns 1 if the item has been recommended at least once, 0 otherwise. Recommender systems with limited coverage can only provide recommendations for a small number of items, posing a significant challenge as the recommender system fails in one of its core functions and limits user exploration. This metric is particularly useful in identifying issues within a recommendation model, such as the cold start problem in Collaborative Filtering algorithms, where items lacking user interactions are unlikely

to be suggested to users.

- **Item Coverage Hit (Item Cov Hit)** It is similar to Item Coverage but this time the items that have received recommendations needs also the have a hit to be considered in the metrics.

$$Hit\ Coverage\ = \frac{1}{|I|} \cdot \sum_{i \in I} \text{hit}(i) > 0 \tag{2.38}$$

Where the function $\text{hit}(i) > 0$ returns 1 if the item has been recommended and has at least one hit, 0 otherwise.

- **Normalized Shannon Entropy (Shannon Div)**

This aggregate diversity metric assesses the distributional dispersion of the recommendation frequencies by considering not only whether an item was recommended but also the frequency of those recommendations. It quantifies the Shannon entropy of the recommendation frequencies for each item.

$$Shannon\ = -\sum_{i \in I} \frac{rec(i)}{rec_t} \cdot \ln \frac{rec(i)}{rec_t} \tag{2.39}$$

Where $rec(i)$ is the number that item $i$ has been recommended and $rec_t$ is the total number of recommendations. In this way $\frac{rec(i)}{rec_t}$ is the frequency of recommendation for item $i$. In contrast to coverage, this metric also takes into account the frequency of item recommendations.

- **Gini Diversity (Gini Div)**

This aggregate diversity metric also relies on the frequency of item recommendations. It is based on the Gini Index definition but has its range flipped in such a way that a higher diversity recommender, therefore with balanced frequencies, will have a low Gini Index but a high Gini Diversity. This formulation is made to facilitate comparisons with other diversity metrics that exhibit a similar behavior. It's important to note that, in this case, item frequencies should be sorted in decreasing order. Function $s(i) = j$ given item $i$ will provide its index $j$ in the original non-sorted data.

$$Gini\ = \sum -i = 1^{|I|} \frac{2i - |I| - 1}{|I|} \cdot \frac{rec(s(i))}{\text{rec}_\text{t}} \tag{2.40}$$

Where $rec(j)$ is 1 if the item has been recommended at least once, 0 otherwise, and $rec_t$ is the total number of recommendations for the test users.

- **Mean Inter-List Diversity (MIL Div)**

  This measure evaluates the dissimilarity of recommendation lists across all users. A comprehensive explanation of how this metric was formulated is available in reference [15][90]. Unlike metrics based on overall item counts, this diversity metric is computed using the actual recommendations received by each user. This metric measures the uniqueness of recommendation lists for individual users, with values ranging from 0 to 1. A higher score signifies an increased probability that two users have received dissimilar recommendations, indicating a higher degree of diversity in the recommendations. The inter-list distance for two users $u_a$ and $u_b$ where $q(u_a, u_b)$ is the number of common items in their recommendation lists:

$$h(u_a, u_b) = 1 - \frac{q(u_a, u_b)}{N} \tag{2.41}$$

  MIL is computed, as an average over all the inter-list distances, excluding the diagonal:

$$MIL_{div} = \frac{1}{|U|^2 - |U|} \sum_{\substack{u_a, u_b \in U \\ u_a \neq u_b}} h(u_a, u_b) \tag{2.42}$$

  This metric demands significant computational resources and, as a result, is only applicable to relatively small datasets. In the evaluation framework, a more efficient formulation of MIL is employed. As demonstrated in reference [15], when dealing with a standard recommender system evaluation scenario, which includes recommendation lists of equal length and no repeated items in these lists, the MIL metric does not necessitate the computation of common items between all possible pairs of users. Instead, it solely relies on the total frequency of recommendations for each item.

$$MIL = 1 - \frac{1}{|U|^2 - |U|} \frac{1}{k} \cdot \left( -|U|k + \sum_{i \in I} \mathrm{rec}(i)^2 \right) \tag{2.43}$$

## 2.5. Reproducibility in Recommender System Research

In numerous recent research papers on recommender systems, there is a common assertion of substantial advancements compared to the existing "state-of-the-art."
In the realm of contemporary recommender systems research, authors have considerable flexibility in the selection of experimental conditions, including the choice of metrics, evaluation protocols, datasets, and baseline models. This leads the authors to present results where their proposed method performs optimally, often without providing a rationale for their choices. Moreover, what these papers often neglect to explicitly mention is that such claims are typically contingent on highly specific experimental conditions.
Over the past few years, there has been a notable surge in the exploration of integrating neural techniques into recommender systems. In particular Deep learning techniques are advantageous when dealing with unstructured data sources and large datasets but it is also one of the main recommended system categories in which these problems are emerging. Of course, they are not restricted to this category only.
Considering the potential methodological issue within our field, we've been wondering to what extent recent neural techniques have truly contributed to advancing the generation of recommendation lists.
Some studies about the reproducibility of new recommendation systems algorithms were done a few years ago [17], we are now wondering if anything has changed in the landscape of recommendation system research. Consequently, we are going to run the same experiment on other more recent studies taken from top conferences.
To perform our experimental work we assess the new proposed algorithms within the same previously employed framework [16], which has unveiled substantial disparities between the reported and the validated outcomes.
Numerous studies have also showcased the ability of older methods to outperform newer algorithms when they are appropriately fine-tuned, as demonstrated in reference [54]. This phenomenon is often attributed to the absence of standardized benchmarks against which new algorithms can be effectively assessed.
Another significant issue related to reproducibility pertains to the common scenario where a paper introducing a new algorithm asserts to offer a precise and comprehensive explanation of his work, but, in reality, falls short in providing adequate information about the technical methodology employed and the setup used for evaluation. Frequently, there are gaps in details, reliance on proprietary data, or disparities between the stated approach and the actual implementation. Additionally, the publication often does not include the link to the source code. Hence, we undertook a comprehensive series of experiments aimed at reproducing the outcomes presented in numerous recent papers advocating neu-

ral recommenders, as published in leading scientific conferences. We then compared these findings with appropriately optimized baseline models executed within the identical environment and under equivalent conditions, as cited in references [16] and [17].
Our primary objective was to validate the feasibility of reproducing the reported results, ensuring the absence of methodological evaluation errors, and gauging the competitiveness against conventional non-neural approaches.

# 3 | Experimental Evaluation Process

In this chapter, we are going to describe the experimental evaluation process we followed in order to conduct our research.
Specifically, we are going to present the method we followed to select the papers to be considered for our reproducibility work along with all the processes we followed to identify the correct paper candidates.
Then we are going to give more details concerning the experimental setup and all the decisions we have taken in order to guarantee the most impartial analysis possible. Describing the methodological approach followed to reproduce the selected algorithms.
Lastly, we are going to present the main components of the framework we used to conduct the systematic and consistent execution of experiments.

This chapter is structured in the following sections:

- *Identifying Relevant and Candidates Papers*
- *Experimental Configuration*
- *Evaluation Framework*

## 3.1. Identifying Relevant and Candidates Papers

This section introduces the studies considered for our research. Initially, we describe the methodological process through which we identified *relevant* and *candidate* for reproduction works, subsequently, we provide a general overview of the chosen studies.

### 3.1.1. Selection Criteria

#### Relevant Papers Definition

A paper needed to meet the following set of criteria to be considered as *relevant*:

- The paper had to be related to the recommender systems domain.
- The paper introduced a novel neural recommendation technique for top-K recommendation either pure Collaborative Filtering or a Hybrid method using precomputed item features. (e.g., methods that are trained on non-structured data such as images or reviews are not included, while methods that use pre-trained embeddings are included). Papers of the following categories are excluded: Cold Start, Sequence Aware, Review-Based, and News Recommendation.
- The paper had to be published between 2021 and 2023.
- The paper had to be published in the proceedings of either the International Joint Conference on Artificial Intelligence (IJCAI) or the World Wide Web (WWW) Conference, workshop papers are not included.

## Candidate Papers Definition

Instead to qualify as *candidate* a *relevant* paper needs to meet the following set of criteria to guarantee we have all the needed information and resources in order to perform our reproducibility experiment:

- The source code must be available in its wholeness. If the source code is not publicly available we instantiated contact with all the authors asking for it.
- At least a dataset needs to be publicly available along with all the necessary information concerning data preprocessing and partitioning.

## Reproducible Papers Definition

Finally, the *reproducible* papers are the ones in which we successfully reproduced the performance results declared in the paper.

## Methodology

To select the *candidate* papers, we adopted a step-by-step approach for each conference. At the time of the selection of the *relevant* papers of this thesis (June 2023), IJCAI 2023 had not yet taken place. So we didn't consider it. We first processed the proceedings of IJCAI 2022 and 2021 and then we moved on with the WWW conferences. WWW is a very big conference and as soon as we got a reasonable amount of *relevant* papers to conduct our experiment we decided not to continue to process the proceedings which was a very time-consuming task, rather we preferred to dive into the reproducibility experi-

ment for the *relevant* papers already selected. As a consequence, we entirely analyzed the proceedings of WWW 2023 and we partially analyzed the proceedings of WWW 2022.
We meticulously reviewed all articles, looking for papers that were aligned with our selection requirements. When a paper met our relevance criteria, as previously described, we proceeded with an in-depth analysis. Subsequently, if the article did not include the source code required to reproduce its experiments, we initiated contact with all of the paper's authors to request access to the code. Upon receiving a response, we noted whether the source code was shared with us or not, along with the reason if it was not provided (e.g., code removal, nondisclosure agreement). If we did not receive a response within a month, we made a second attempt to contact the authors. In the event that the second attempt was unsuccessful, we recorded the absence of a response.
Among the *relevant* papers, once we had acquired the source code, either from the article itself or through communication with the authors via email, we classified a paper as *candidate* if it included the entire algorithm model, encompassing its training process, and the code necessary for generating item predictions. Papers that provided only a partial version of the algorithm were not considered *candidates*, while papers that had issues with compilation and execution in the source code were considered *candidate* papers with non-executable source code and therefore not *reproducible*.
A final criterion for classifying a *relevant* paper as a *candidate* was the availability of at least one of the datasets used for evaluation in the original paper. In certain cases, the authors also furnished the data partitions (train, validation, and test) employed in their experiments. If these data partitions were not provided, we deemed papers as *candidate* only if they included comprehensive information concerning data preprocessing and partitioning, ensuring the same experimental conditions as those used by the authors.
Once we defined which were the *candidate* papers we started to reproduce the algorithms. In this work, to conduct our reproducibility experiment we are going to follow the reproducibility guidelines of the Association for Computing Machinery[1] (ACM).

### 3.1.2. Relevants Papers

As we outlined in Section 3.1, we conducted a search for *relevants* papers in the IJCAI and WWW conferences ranging between the years 2021 and 2023. This section provides an overview of the *relevant* papers for each conference and specifies which of them meet our criteria for being considered *candidates* for reproduction. Table 3.1 groups the IJCAI 2021 and 2022 *relevant* papers while Table 3.2 groups the WWW 2022 and 2023 *relevant* papers.

[1] `https://www.acm.org/publications/policies/artifact-review-and-badging-current`

| Year | Title | Candidate | Citation |
|---|---|---|---|
| 2022 | Self-supervised Graph Neural Networks for Multi-behavior Recommendation | No | Gu et al. [21] |
| 2022 | RecipeRec: A Heterogeneous Graph Learning Model for Recipe Recommendation | Yes | Tian et al. [66] |
| 2022 | Trading Hard Negatives and True Negatives: A Debiased Contrastive Collaborative Filtering Approach | No | Yang et al. [82] |

Table 3.1: List of papers in the proceedings of IJCAI 2022 and 2021 which are identified as *relevants*.

In Table 3.1, there is no paper selected from the IJCAI 2021 conference as we analyzed the proceeding related to recommendation systems and found that no papers met our selection criteria.

In [21] the source code was not publicly available. We tried to contact the authors asking if they could provide us with the code but no answer was given. A second attempt to contact them was made 4 weeks later but without any success. We therefore marked the paper as not a *candidate* for reproduction.

Also in [82] the source code was not publicly available. We initiated contact with the authors asking for the source code but they informed us that the codes of their paper were unavailable. That was due to a positional shift of the main author from an internship to a laboratory and as a result of this shift, the codes were not transferred to his new position.

| Year | Title | Candidate | Citation |
|---|---|---|---|
| 2023 | Automated Self-Supervised Learning for Recommendation | Yes | Xia et al. [81] |
| 2023 | Bootstrap Latent Representations for Multi-modal Recommendation | Yes | Zhou et al. [91] |
| 2023 | Multi-Modal Self-Supervised Learning for Recommendation | Yes | Wei et al. [73] |
| 2023 | Multi-Behavior Recommendation with Cascading Graph Convolution Networks | Yes | Cheng et al. [10] |
| 2023 | Personalized Graph Signal Processing for Collaborative Filtering | No | Liu et al. [38] |
| 2022 | Fast Variational AutoEncoder with Inverted Multi-Index for Collaborative Filtering | Yes | Chen et al. [7] |

Table 3.2: List of papers in the proceedings of WWW 2023 and 2022 which are identified as *candidates*.

In [38] The source code was not publicly available. We tried to contact the authors asking if they could provide us with the code. They positively replied saying that they made publicly available the source code through a GitHub link but unfortunately, after some time they removed the code from the platform leaving us with a not working link.

To reach this number of six *relevant* papers we screened the papers published at the conferences taken into the study and then we analyzed in more detail almost 50 papers looking if they were in line with our requirements.

### 3.1.3. Candidate Papers for Reproduction with Non-Executable Source Code

We classified as *candidates* but non-executable all those papers in which we tried to conduct our experiment but, it was not possible to port the models successfully in our evaluation framework due to the following issues:

- The source code is not clear in its structure, leading us to make an unreasonable effort to complete the porting in our framework.
- The original implementation leverages some libraries that are no longer supported (i.e., deprecated libraries). Making us unable to reproduce the algorithm.

Table 3.3 summarizes those papers that belong to this category.

| Year | Title | Citation |
|---|---|---|
| 2022 | RecipeRec: A Heterogeneous Graph Learning Model for Recipe Recommendation | Tian et al. [66] |
| 2023 | Multi-Behavior Recommendation with Cascading Graph Convolution Networks | Cheng et al. [10] |

Table 3.3: List of papers classified as *candidates* for which we didn't manage to successfully execute the experiment.

We provide a concise overview of the paper's content along with the rationale behind classifying it as non-executable:

- *RecipeRec: A Heterogeneous Graph Learning Model for Recipe Recommendation:* This paper was about recipe recommendation, the authors formalize the problem of recipe recommendation with graphs to incorporate the collaborative signal into recipe recommendation through graph modeling. In particular, they first present URI-Graph, a new and large-scale user-recipe-ingredient graph. Then they propose RecipeRec, a novel heterogeneous graph learning model for recipe recommendation. The original implementation was entirely made on a Python Jupyter Notebook[2] leveraging the dgl[3] Python library. The code was not particularly clear in its parts. After two weeks of reasonable effort working on this algorithm, we realized that a complete refactoring of the code was needed to correctly reproduce the work, however, this rewrite is not compatible with a reproducibility study since this kind of work to be properly conducted involves the use of original artifacts as outlined in the guidelines of the ACM[4].
  As a consequence, we decided not to undertake such an invasive rearrangement so as not to invalidate the scope of our research, and to keep the integrity of the original implementation. Therefore this algorithm was marked as non-executable in our framework.

- *Multi-Behavior Recommendation with Cascading Graph Convolution Networks:* In this work, the authors propose a novel multi-behavior recommendation model with cascading graph convolution networks named MBCGCN. In MBCGCN, the embeddings learned from one behavior are used as the input features for the next behav-

[2] `https://jupyter.org/`
[3] `https://www.dgl.ai/`
[4] `https://www.acm.org/publications/policies/artifact-review-and-badging-current`

ior's embedding learning after a feature transformation operation. In this way, their model explicitly utilizes the behavior dependencies in embedding learning.
The original implementation was entirely made with Python 3.6 leveraging TensorFlow[5] 1.14 which was released in 2019. This version is now deprecated, and no longer supported, in favor of new releases such as the last one which is TensorFlow 2.14. It was already obsolete when the paper was published, and now it is incompatible with other necessary libraries. Publishing source that uses outdated versions is a bad practice as it makes it very difficult to reproduce the experimental setup.
We weren't able to run our experiment with the original code due to the incompatibility of libraries, therefore we marked this work as non-executable.

### 3.1.4. Reproducibility Outcome

In this section, we highlight and give a general overview of the papers for which the experiment has been completed. In particular for those papers that have been revealed as *reproducible* namely papers for which we were able to reach the performance values stated in their original work.
Table 3.4 shows which papers are classified as *reproducible* among those that were both executable and *candidates*.

| Year | Title | Reproducible | Citation |
|---|---|---|---|
| 2023 | Automated Self-Supervised Learning for Recommendation | Yes | Xia et al. [81] |
| 2023 | Bootstrap Latent Representations for Multi-modal Recommendation | Yes | Zhou et al. [91] |
| 2023 | Multi-Modal Self-Supervised Learning for Recommendation | Yes | Wei et al. [73] |
| 2022 | Fast Variational AutoEncoder with Inverted Multi-Index for Collaborative Filtering | No | Chen et al. [7] |

Table 3.4: List of executable papers along with their reproducibility outcomes according to our criteria.

The papers that are *reproducible* are:

- *Automated Self-Supervised Learning for Recommendation:* This work proposes a unified Automated Collaborative Filtering (AutoCF) to automatically perform data

[5] `https://www.tensorflow.org/`

augmentation for recommendation. Specifically, the authors focus on the generative self-supervised learning framework with a learnable augmentation paradigm that benefits the automated distillation of important self-supervised signals. To enhance the representation discrimination ability, their masked graph autoencoder is designed to aggregate global information during the augmentation via reconstructing the masked subgraph structures.

- *Bootstrap Latent Representations for Multi-modal Recommendation:* This paper studies the multi-modal recommendation problem, where the item multi-modality information (e.g., images and textual descriptions) is exploited to improve the recommendation accuracy. The authors propose a novel self-supervised multi-modal recommendation model (BM3), which requires neither augmentations from auxiliary graphs nor negative samples. BM3 tries to alleviate both the need for contrasting with negative examples and the complex graph augmentation from an additional target network for contrastive view generation.

- *Multi-Modal Self-Supervised Learning for Recommendation:* The online emergence of multi-modal sharing platforms (e.g., TikTok, Youtube) is forcing personalized recommender systems to incorporate various modalities (e.g., visual, textual, and acoustic) into the latent user representations. Inspired by the recent progress of self-supervised learning in alleviating label scarcity issue, the authors propose a new Multi-Modal Self-Supervised Learning (MMSSL) method that tackles some important challenges.

The papers that are non-reproducible are:

- *Fast Variational AutoEncoder with Inverted Multi-Index for Collaborative Filtering:* Variational AutoEncoder (VAE) has been extended as a representative nonlinear method for collaborative filtering. However, the bottleneck of VAE lies in the softmax computation over all items, such that it takes linear costs in the number of items to compute the loss and gradient for optimization. This hinders the practical use due to millions of items in real-world scenarios. To this end, the authors propose a fast Variational AutoEncoder (FastVAE) for collaborative filtering.

A complete and exhaustive analysis for each experiment is available in Chapter 4.

## 3.2. Experimental Configuration

The experimental setup is based on a process of adapting algorithms from the selected papers to a standardized framework, distinct from their original environments. The objec-

tive is to ensure that all considered algorithms are evaluated under the same conditions. To maintain the utmost fairness in our analyses and comparisons, we decided to reproduce the experiments exactly as originally conducted by the model authors. To achieve this, we retained several key elements in their original form, including datasets, the model itself, optimal hyperparameters, and evaluation metrics. We kept the core algorithm implementations unaltered to avoid potential discrepancies when comparing our results with those obtained by the authors. As the first step, we implemented, for each dataset utilized in the experiments, the code part in which data is loaded. We applied the same data splitting procedure as found in the original code when available. Otherwise, we applied the preprocessing and splitting steps as described in the paper on the original dataset and then we created the URM / ICM sparse matrices needed to properly train the model. Secondly, we wrapped the original model in our RecommanderWrapper class which encapsulated all the procedures for fitting the recommender model, saving and loading the model, and computing item recommendations for users. After the computation of item scores, they were directed to a dedicated function within our framework responsible for the evaluation process. This function organizes the scores to produce the ultimate recommendation list and computes the evaluation metrics. This methodology guaranteed that the outcomes were exclusively affected by the recommender model's performance, effectively removing any possible sources of bias arising from how metrics were computed or the sorting algorithm employed. Our evaluation includes baseline models and metrics. A Bayesian search approach is used to determine optimal hyperparameters. This approach was used both for tuning the baselines and for enhancing recommendation quality. Whenever possible, we used the same hyperparameters reported in the original paper and selected the number of training epochs through early stopping. To analyze the model's computational complexity and scalability we took into account the training time and the recommendations per second generated by each algorithm, during the experimental phase.

### 3.2.1. Baselines and Baselines Hyperparameter Optimization

To achieve a comprehensive perspective, we decided to adopt different categories of baseline recommender algorithms in our assessments in order to compare the algorithm's results to well-established methods. All of them are described in Chapter 2. Table 3.5 provides an overview of all the baseline models employed.

| Family | Method | Description |
|---|---|---|
| Non-Personalized | TopPopular | Recommends the most popular items. |
| Item Based CF | ItemKNN CF cosine | Item-based k-nearest neighbors. |
| Graph-Based | RP3beta | Extension of *P3alpha*. |
| | GraphFilter CF | Graph-based collaborative filtering method. |
| Matrix Factorization | MF BPR | Matrix factorization model with BPR. |
| | PureSVD | Basic matrix factorization method. |

Table 3.5: List of Baselines Employed.

Selecting the most suitable hyperparameters for the learning algorithms is a critical task as it can significantly impact a model's performance. We first split the dataset into training validation and test sets. Then we automated the tuning of hyperparameters for the baseline algorithms by employing a Bayesian search approach on the validation set, implemented using Scikit-Optimize version 6. The hyperparameter ranges in which the hyperparameter tuning is performed are shown in Table A.32. Hyperparameter optimization is exclusively conducted on the baseline algorithms, and not on the recommender models. This decision is based on the frequent availability of hyperparameter information reported by the authors in the original paper or in the source code. Additionally, fine-tuning the hyperparameters of deep learning models often requires a huge computational effort, making it a less practical activity. After establishing the optimal hyperparameters through the evaluation of the model using validation data, the ultimate model was trained on the combined dataset of the union of training and validation sets. Subsequently, its performance was assessed using the test data.

### 3.2.2. Early Stopping

Almost all machine learning models are subjected to training for a number of epochs, during which the model's performance initially improves before reaching a stable phase, often characterized by a degree of variation. Consequently, the selection and optimization of the number of epochs is another critical hyperparameter. It's crucial to emphasize that in the majority of the papers under analysis, neither the number of training epochs nor the termination criteria are explicitly specified. Moreover, when examining the source code provided by the authors, in certain cases, the number of training epochs seems to be determined through the evaluation of the test set rather than the validation set, leading to the overfitting of the model. Early stopping is a widely adopted method for determining the ideal number of training epochs. The concept behind early stopping

involves periodic evaluation of the model's performance on the validation data during the training process. The training is stopped when, over a specified number of validation steps, the model's quality fails to surpass the best solution achieved thus far. Early stopping offers several advantages, including the use of precise and transparent criteria to select the number of epochs. This approach eliminates the need for arbitrary manual optimization and frequently leads to shorter training durations avoiding overfitting. In our work, we applied the early stopping method to both the baseline models and the deep learning models. Given the time-intensive nature of the evaluation process, we conducted the validation step at intervals of every five training epochs. If, for a continuous sequence of twenty-five validation steps, the recommendation quality of the current model does not improve compared to the best result achieved up to that point, the training process is stopped as shown in Figure 3.1.

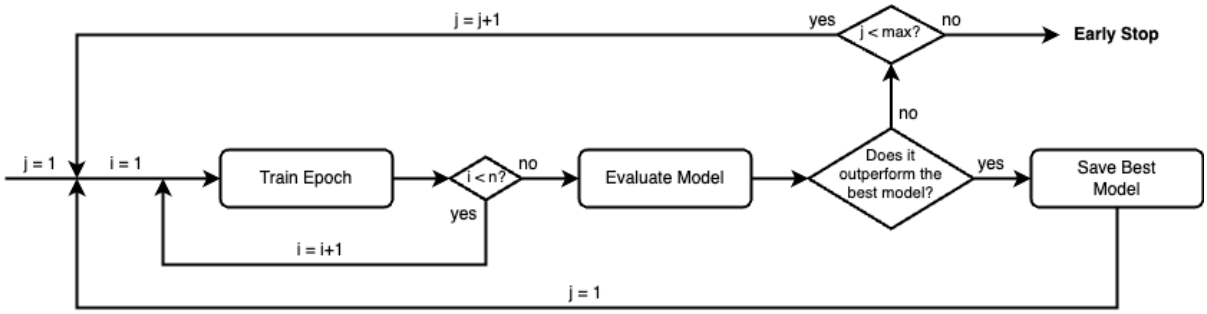


Figure 3.1: Early stopping training flow chart. Where n is the number of epochs to run before each validation step (in our framework, it is set to 5) and max is the number of non-improving validation steps to stop the training with early stopping (in our framework, it is set to 25). While i and j are the counters to be compared with the threshold n and max respectively.

### 3.2.3. Training Time and Recommendation Time

When we talk about algorithms a crucial aspect is the the computational cost in terms of time. Recommender systems are not an exception. Specifically, in the context of deep learning-based recommender systems, having reasonable training and recommendation time is imperative to ensure robust scalability when dealing with extensive datasets in a real-world production setting. To address this aspect within our framework, we examine three distinct metrics for both the baseline methods and the analyzed algorithms. We provide information on the *Train Time*, representing the total duration of the model training, the *Recommendation Time*, which quantifies the time necessary for generating recommendation lists during model evaluation, and finally, the *Recommendation Throughput*, measuring the system's ability to deliver recommendations in terms of users served

per second. It's essential to acknowledge that several external factors, distinct from the algorithm, can impact both the training and recommendation time. These factors encompass the execution platform, the processing unit's workload during execution, and the code's optimization. Hence, our baseline algorithms are constructed either by leveraging the native NumPy functions and vectorization within Python or by creating a specialized Cython module for the most computationally demanding tasks. In contrast, the algorithms we reproduce are maintained in their original implementations.

## 3.3. Evaluation Framework

To ensure the systematic and consistent execution of experiments, we employed the evaluation framework used in similar previous works [17]. This framework is based on an object-oriented paradigm that unifies all algorithms within a common structure, achieved through a hierarchical system of classes and their corresponding methods. This design is meticulously structured to prevent errors and maintain result consistency across all experiments. Throughout our experiment we assumed the correctness of the primary code used by the original authors to build the model, considering the code as the final version they used for their experiments and their reported results. Consequently, in our experiments, we exclusively utilized the authors' original model code, adapting it as necessary to align with the structure of our framework. In the next sections, we are going to present all the core classes that are responsible for the execution of the experiment.

### 3.3.1. DataReader

The *DataReader* class is responsible of everything that concerns data loading, data preprocessing, and data splitting. The data is inserted into a dictionary structure to load it in a faster way during multiple execution of the experiment. In the initialization method in fact, the class first attempts to load a previously saved data version. If no such version exists, it proceeds to parse the original data, applying any necessary preprocessing. The result is a dictionary that includes the User-Item Interaction Matrix (URM), Item-Content Matrix (ICM), User-Content Matrix (UCM), and all essential data structures required for the correct execution of both the baseline and the method outlined in the research paper. If the original authors provided a train-test-validation split, we utilized it by converting the original structures into sparse matrices to ensure compatibility with the baseline algorithms. Following the creation or loading of the data structures, we verified their disjoint nature and, in certain cases, confirmed if they properly contained implicit data, only for those algorithms that relied on it. A distinct *DataReader* class was established

for each algorithm under analysis.

### 3.3.2. BaseRecommender

The *BaseRecommender* class takes on the responsibility of training and optimizing the model. It includes the necessary classes for constructing and adapting all recommender models, offering the following abstract methods:

- `fit`: this method is responsible for fitting the model.
- `_compute_item_score`: it provides the item scores for the designated user and a set of items. The original implementation is used whenever is available.
- `recommend`: this function returns recommendations for the users, along with their associated item scores, after applying some filtering, such as excluding items the user has already interacted with. This function invokes the `_compute_item_score` method to compute item scores. In the end, the function delivers recommendations to the user, selecting items with the highest scores.
- `save_model`: it saves the trained model with all its needed parameters into a file given as an argument.
- `load_model`: it restores a previously saved model.

All recommender models, including both the baseline and deep learning algorithms utilized in the experiments, inherit these methods and implement them. This inheritance is the reason why these methods are consistently available across these models.

### 3.3.3. Evaluator

The Evaluator class evaluates a given recommender model and computes the list of metrics. Its methods are:

- `init`: it initializes the evaluator with a test dataset and a list of cutoff values for metric evaluation.
- `evaluateRecommender`: it assesses the provided trained recommender object by calculating the metric values for each specified cutoff.

### 3.3.4. Incremental_Training_Early_Stopping

The *Incremental_Training_Early_Stopping* class trains the recommender model using the early-stopping procedure. It implements the following methods:

- `_run_epoch`: this method contains all the source code, provided by the original implementation, needed to run each single epoch.
- `_train_with_early_stopping`: this method handles both the training and validation processes. To execute it, you need to provide parameters including the minimum and maximum number of training epochs, the interval at which the model should be evaluated with the validation set (5 in our case), a boolean to determine whether early stopping should be performed or not, the designed metric used to determine the best model, and the evaluator instance used to compute validation metrics.
- `_update_best_model`: it just calls the `save_model` method when the current model performs better on the validation set than the previously saved one.
- `_prepare_model_for_validation`: method called before the evaluation phase, just after finishing the training of the model. It contains any procedure needed before evaluating the performance of the algorithm.

To provide a general overview of the execution process, initially, the model is created within the `fit` method of the *BaseRecommender* class. Subsequently, the loop initiates by invoking the `_train_with_early_stopping` method, which conducts the training steps for each epoch. These steps involve training the model incrementally, evaluating its performance on the validation set at specified intervals using the `evaluateRecommender` method from the *Evaluator* class, and potentially updating the best model if an improvement is identified.

### 3.3.5. RecommenderWrapper

As we had to use the authors' original source code, it was essential to wrap the code into our evaluation framework. To perform this task, we used a *RecommenderWrapper* class specifically created for each model (in the code base this class is referred to as modelname_RecommenderWrapper). This adaptation permits us to run all the algorithms in the same way, even when they are written with distinct conventions and rely on different machine-learning libraries, such as Tensorflow and Keras. In order to correctly wrap the models we had to let the *RecommenderWrapper* classes of the paper algorithms inherit from three classes: *BaseRecommender*, *IncrementalTrainEarlyStopping*, and *BaseTempFolder* useful if the algorithm necessitates saving some temporary file. Figure 3.2 illustrates the inheritance diagram.

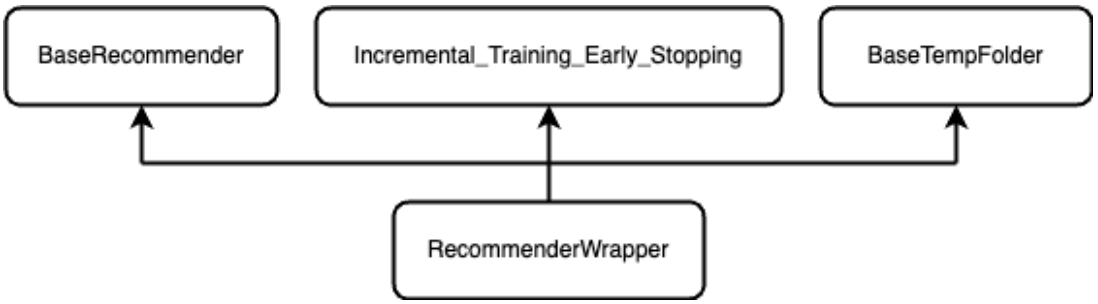


Figure 3.2: Inheritance diagram of the *RecommenderWrapper* class used to wrap the reproduced algorithms.

### 3.3.6. Experiment Execution

To conduct a robust and complete experiment we decided to run each algorithm two times. Each of the two runs is composed of two training steps. In the first run, we train the model without early stopping on the train set with the train-validation-test split provided. The model is then trained again on the union of train and validation sets to bring out the maximum performance of the model leveraging all the available data.
In the second run, we train the model with early stopping on the train set using the train-validation-test split, when the training ends we save the number of epochs executed to get the best model and then we train the model again this time without early stopping only for the selected number of epochs on the union of train and validation sets. These execution patterns allow us to generate the best performance possible from each algorithm and avoid any possible doubt about the values obtained. All the models after training are then evaluated on the test set, which is never used outside this scope. In the end, the best model for the NO early stopping and for the early stopping executions is saved. Figure 3.3 shows the process described.

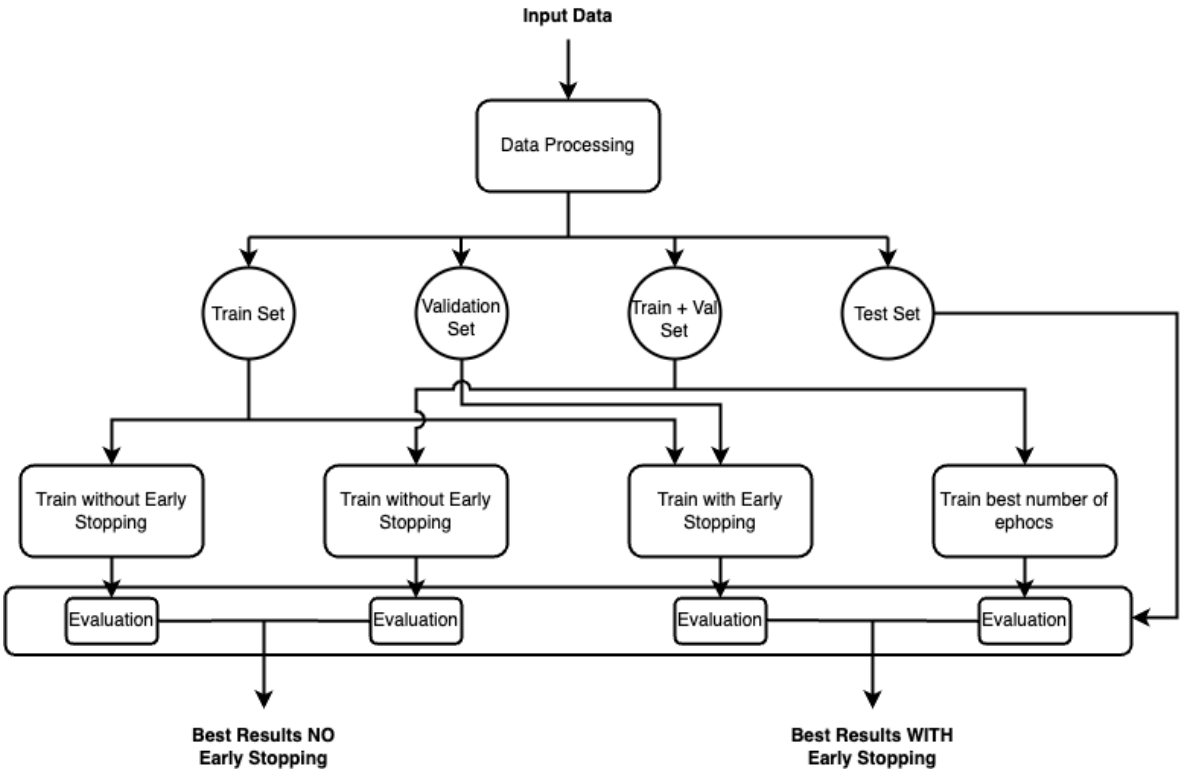


Figure 3.3: Schema showing the execution of each algorithm's experiments.

# 4 | Analysis and Results of Reproducible Papers

In this chapter, we are going to provide the analysis and results of the reproduced algorithms. For each method, we are going to provide:

- Introduction to the method.
- Analysis of the Dataset.
- Brief description of the original evaluation protocol.
- The overview of the paper baselines.
- Description of the hyperparameters of the method.
- Discussion and results of the experiment.

In our experimental work, we trained more than 3000 models considering the reproduced algorithms and the baselines with their hyperparameter tuning spending around 18 days of computational time. Full results for each method on each dataset are reported in Appendix A

## 4.1. Automated Self-Supervised Learning for Recommendation

In [81], published at World Wide Web Conference 2023 (WWW23), the authors propose a new Automated Collaborative Filtering (AutoCF) framework to automatically perform self-supervised augmentation in graph-based CF paradigm. They claim that their framework is capable of distilling the graph structure-adaptive self-supervised signals for advancing the graph-based CF framework. In particular, they design a learnable masking function to automatically identify the important centric nodes for reconstruction-based data augmentation. In the mask learning stage, the node-specific subgraph semantic relatedness is considered to accurately preserve the graph-based collaborative relationships.

Additionally, they propose a new masked graph autoencoder in which the key ingredient is a graph neural encoder that captures the global collaborative relationships for reconstructing the masked user-item subgraph structures. The framework is composed of three different stages:

1. The adaptive graph structure augmentation based on infomax relatedness
2. The masked graph autoencoder based on the graph self-attention architecture
3. The self-augmented training paradigm includes graph structure reconstruction, node-subgraph mutual information maximization, and contrastive learning.

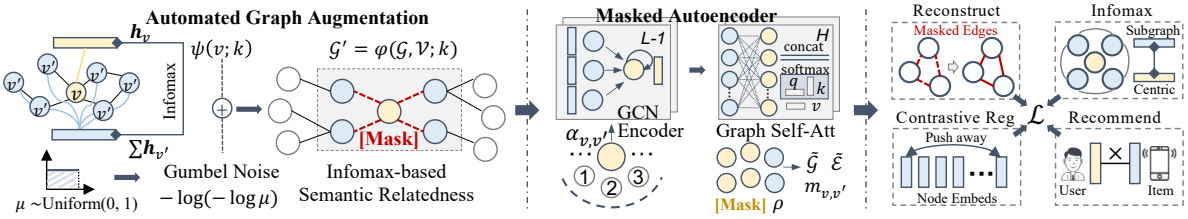


Figure 4.1: The architecture of the proposed framework Automated Collaborative Filtering (AutoCF)

To summarize, their main contributions are:

- Investigate the drawbacks of existing contrastive GNN-based recommendation methods with the non-adaptive self-supervised augmentation and weak robustness against noise perturbation.
- Propose an automated self-supervised learning model AutoCF, in which a learning to mask paradigm is designed to perform data augmentation with structure-adaptive self-supervision. In addition, the automated mask generator is integrated with graph masked autoencoder to enhance user representations.

### 4.1.1. Datasets

In order to increase the difficulty of the recommendation task they utilized three popular datasets: Amazon, Yelp and Gowalla. The evaluation is performed on the datasets using the split available in the GitHub repository. Their statistics are reported in Table 4.1. All existing interactions are made implicit and assigned a value of 1. The characteristics size and density of the datasets used for the reproducibility experiments are consistent with the ones reported in the original article.

**Gowalla:** This dataset is collected from a location-based service to record the check-in behaviors between users and different locations from Jan 2010 to Jun 2010.

**Amazon:** This is the Amazon book category dataset. The version used is the one with implicit feedback. If the user has saved the article in their library it will be associated with a rating of 1.

**Yelp:** Yelp is an American platform that publishes crowd-sourced reviews about business. The dataset contains user-venue rating interaction from Jan 2018 to Jun 2018.

| Dataset | Interactions | Items | Users | Sparsity |
|---|---|---|---|---|
| Amazon | 1380972 | 83761 | 76469 | 9.998E-01 |
| Gowalla | 421405 | 19747 | 25557 | 9.992E-01 |
| Yelp | 260510 | 26822 | 42712 | 9.998E-01 |

Table 4.1: AutoCF Dataset characteristics

### 4.1.2. Evaluation protocol

The data splitting is performed in the following way: 70% of the observed interactions are randomly sampled to generate the training set. The remaining 30% data, 5% and 25% percent of interactions are used for validation and testing. They adopted the all-rank evaluation protocol [31] to measure the item recommendation accuracy for each user following the same settings in [79][80]. They used two representative evaluation metrics Recall and NDCG at cutoffs 20, and 40 to evaluate all methods. The original train-test split of the data is available on the Github repository. In order to keep the results presented in this report consistent, we used the dataset already preprocessed given in the original source code. Looking at the dataset statistics we can notice that hard preprocessing is made in order to make those datasets more dense and reduced in size, especially for the Amazon one in which the original version has six million users. Unfortunately, no information on how they are processed is given.

### 4.1.3. Baselines

The paper reports the following 14 baselines belonging to 5 different categories:

**1) Conventional Collaborative Filtering Methods**

- **BiasMF** [30]: This is a widely adopted matrix factorization baseline which projects users and items into latent vectors.
- **NCF** [24]: It is a representative neural CF method that replaces the inner product

with the non-linear feature projection.

**2) Autoencoder-based Recommender System**

- **AutoR** [59]: It employs the autoencoder to generate the embeddings of users and items with the reconstruction loss.

**3) GNN-based Collaborative Filtering**

- **PinSage** [86]: It leverages the graph convolution network to model the user-item interaction through graph structures.
- **STGCN** [87]: It design the graph autoencoding for high-order interaction learning with the reconstruction regularization.
- **GCMC** [67]: It is one of the pioneering CF models that enhances the matrix completion with GNN-based message passing.
- **NGCF** [71]: It is one of the state-of-the-art graph CF models that incorporates high-order connectivity for recommendation.
- **GCCF** [8]: To simplify the message passing scheme, recent studies propose to omit the non-linear feature transformation and activation, which achieves better results.
- **LightGCN** [25]: a Light model applying graph convolution for collaborative filtering.

**4) Disentangled Representation-enhanced GNN Model**

- **DGCF** [72]: This method disentangles latent factors behind user-item interactions under the graph neural network architecture, to capture the fine-grained user-item relationships.

**5) State-of-the-art Self-Supervised Recommendation Methods**

- **SLRec** [84]: This self-supervised learning method conducts data augmentation with the learning of latent feature correlations.
- **NCL** [37]: It enhances the graph contrastive learning with the augmented structural and semantic-relevant training pairs.
- **SGL** [79]: This method applies random structure augmentation such as node/edge drop and random walk, to generate contrastive views for graph augmentation in collaborative filtering. This model contrasts different node views that are generated by randomly masking the edge connections on the graph and incorporate the proposed self-supervised loss into LightGCN.

- **HCCF** [80]: This method conducts local-global cross-view contrastive learning with hypergraph-based structure learning.

### 4.1.4. Hyperparameters

They used the Adam optimizer for parameter inference, with a learning rate of $1e^{-3}$ and batch size of 4096. By default, the hidden dimensionality is set as 32, and the number of graph neural iterations is tuned from {1,2,3}. The graph self-attention employs 4 attention heads. They stated that they conducted subgraph sampling for every 10, 30, or 60 steps to further improve the model efficiency. In their learning to mask paradigm, the size of centric nodes is selected from $\{200, 400, ..., 1800, 2000\}$. The ratio of sampled nodes $\rho$ is set as 0.2. They select the number of graph hops for subgraph masking from {1,2,3} to reflect the high-order semantic relatedness. They reported that the best performance is achieved with 2 hops. In the model training phase, $\lambda_1$ and $\lambda_2$ are respectively chosen from $\{1e^{-k}| -1 \leq k \leq 4\}$ and $\{1e^{-k}|3 \leq k \leq 8\}$ to control the regularization strength of weight-decay factor and the augmented self-supervised signals.
The tuning strategy and criteria to select the number of epochs is not mentioned.
Regarding the baselines, it is not clear whether hyperparameter optimization was performed. However, the embedding size is a hyperparameter to be tuned for each dataset and for each embedding-based algorithm, as different models with different objective functions or training procedures will probably require different values for it. The paper does not describe which metric is optimized so we decided to optimize NDCG@20.
The hyperparameters used for the reproduced method are reported in Table 4.2. They are the default hyperparameters provided by the authors in the code base.

| Hyperparameter | Described in | Value for all datasets |
|---|---|---|
| Regularization rate | Paper | 0.01 |
| Learning rate | Paper | 1e-3 |
| Embedding Size | Paper | 32 |
| Self-Attention Heads | Paper | 4 |
| Batch size | Paper | 4096 |
| Epochs | Source code | 100 |
| Ratio of Sampled Nodes $\rho$ | Paper | 0.2 |
| Subgraph Masking Hops | Source code | 2 |
| $\lambda_1$ | Source code | 1 |
| $\lambda_2$ | Source code | 1e-7 |

Table 4.2: AutoCF Hyperparameters values

### 4.1.5. Results and Discussion

The source code is publicly available on GitHub platform[1]. Tables 4.3, 4.4 and 4.5 report the results contained in the original paper and the results obtained running the ported version of the algorithm when evaluated with the evaluator object of our framework.
In order to validate the results presented in the paper, we use the provided training-validation-test data split for all the datasets.
We were able to reproduce consistent results with the ones declared in the original work for Amazon and Gowalla datasets. In some cases, we got even better results than the original ones.
Instead, we couldn't confirm the original results for the Yelp dataset probably because of non-optimal hyperparameters. Unfortunately is not stated in the paper which are the best hyperparameters used by the authors.
In our experiments, we decided not to run hyperparameter tuning due to the high computational resources needed.
A critical aspect of this algorithm is that in all the datasets, AutoCF is outperformed by the traditional ItemKNN, RP3beta, and GraphFilter CF approaches.
In the Appendix A.0.1 we display the full results for all metrics at different cutoffs.
From the analysis of the beyond accuracy, we can further notice that baselines outperform this method also on its capability to offer diversified recommendations. As we can notice from Table 4.6 taking the Amazon dataset as a reference we can figure out that all the baseline that outperforms the model in terms of recommendation performance also overperform it on the beyond accuracy metrics.
As a last analysis, we can compare the training time and recommendation time. In terms of training time AutoCF took days to be trained instead of minutes for the baselines while in recommendation time it is slightly better than the baselines even if the difference doesn't justify the improvement. The beyond accuracy metrics and time analyses of Gowalla and Yelp datasets can be found in Appendix A.0.1.

[1] `https://github.com/HKUDS/AutoCF`

Table 4.3: Experimental results for the AutoCF method on the Amazon dataset.

| | @ 20 | | @ 40 | |
|---|---|---|---|---|
| | REC | NDCG | REC | NDCG |
| TopPop | 0.0122 | 0.0079 | 0.0199 | 0.0103 |
| ItemKNN CF cosine | **0.1735** | **0.1381** | **0.2148** | **0.1508** |
| RP3beta | **0.1777** | **0.1403** | **0.2220** | **0.1539** |
| GraphFilter CF | **0.1387** | **0.1019** | **0.1827** | **0.1155** |
| MF BPR | 0.0835 | 0.0624 | 0.1126 | 0.0715 |
| PureSVD | 0.0781 | 0.0619 | 0.1055 | 0.0703 |
| AutoCF paper | 0.1277 | 0.0879 | 0.1782 | 0.1048 |
| AutoCF | 0.1312 | 0.0959 | 0.1792 | 0.1107 |
| AutoCF no earlystopping | 0.1315 | 0.0947 | 0.1785 | 0.1093 |

Table 4.4: Experimental results for the AutoCF method on the Gowalla dataset.

| | @ 20 | | @ 40 | |
|---|---|---|---|---|
| | REC | NDCG | REC | NDCG |
| TopPop | 0.0327 | 0.0215 | 0.0473 | 0.0257 |
| ItemKNN CF cosine | **0.2640** | **0.1771** | 0.3595 | **0.2039** |
| RP3beta | **0.2795** | **0.1878** | **0.3778** | **0.2157** |
| GraphFilter CF | **0.2677** | 0.1716 | **0.3688** | 0.2004 |
| MF BPR | 0.2074 | 0.1350 | 0.2871 | 0.1576 |
| PureSVD | 0.1569 | 0.1062 | 0.2236 | 0.1252 |
| AutoCF paper | 0.2538 | 0.1645 | 0.3441 | 0.1898 |
| AutoCF | 0.2629 | 0.1729 | 0.3621 | 0.2009 |
| AutoCF no earlystopping | 0.2610 | 0.1742 | 0.3569 | 0.2013 |

Table 4.5: Experimental results for the AutoCF method on the Yelp dataset.

| | @ 20 | | @ 40 | |
|---|---|---|---|---|
| | REC | NDCG | REC | NDCG |
| TopPop | 0.0229 | 0.0108 | 0.0348 | 0.0137 |
| ItemKNN CF cosine | **0.0753** | **0.0402** | 0.1082 | **0.0484** |
| RP3beta | 0.0715 | **0.0379** | 0.1030 | 0.0457 |
| GraphFilter CF | **0.0759** | **0.0375** | **0.1160** | **0.0473** |
| MF BPR | 0.0371 | 0.0189 | 0.0561 | 0.0236 |
| PureSVD | 0.0521 | 0.0272 | 0.0803 | 0.0341 |
| AutoCF paper | **0.0869** | **0.0437** | **0.1273** | **0.0533** |
| AutoCF | 0.0731 | 0.0364 | 0.1114 | 0.0458 |
| AutoCF no earlystopping | 0.0563 | 0.0283 | 0.0846 | 0.0353 |

Table 4.6: Experimental results for the AutoCF method on the Amazon dataset on beyond accuracy metrics.

| | @ 20 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0025 | 0.0181 | 0.0005 | 0.0004 | 0.0003 | 4.3771 |
| ItemKNN CF cosine | **0.0037** | **0.9979** | **0.9685** | **0.2200** | **0.3653** | **14.9918** |
| RP3beta | **0.0037** | **0.9978** | **0.9638** | **0.2247** | **0.3765** | **14.9953** |
| GraphFilter CF | **0.0040** | **0.9981** | **0.9467** | **0.2009** | **0.4992** | **15.5244** |
| MF BPR | 0.0034 | 0.9927 | **0.6352** | 0.0911 | 0.1319 | 13.3807 |
| PureSVD | 0.0031 | 0.9780 | 0.0825 | 0.0465 | 0.0163 | 10.7424 |
| AutoCF paper | - | - | - | - | - | - |
| AutoCF | 0.0035 | 0.9975 | 0.4645 | 0.1652 | 0.1590 | 14.0278 |
| AutoCF no earlystopping | 0.0035 | 0.9971 | 0.4623 | 0.1651 | 0.1554 | 13.9727 |

Table 4.7: Computation time for the algorithms in the selected results for the AutoCF method on the Amazon dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.03 [sec] | 323.05 [sec] / 5.38 [min] | 220 |
| ItemKNN CF cosine | 54.14 ± 4.79 [sec] | 294.26 [sec] / 4.90 ± 0.22 [min] | 238 |
| RP3beta | 103.80 [sec] / 1.73 ± 0.32 [min] | 292.68 [sec] / 4.88 ± 0.18 [min] | 235 |
| GraphFilter CF | 519.94 [sec] / 8.67 ± 2.74 [min] | 326.61 [sec] / 5.44 ± 0.45 [min] | 233 |
| MF BPR | 8355.98 [sec] / 2.32 ± 1.95 [hour] | 276.03 [sec] / 4.60 ± 0.97 [min] | 348 |
| PureSVD | 23.54 ± 12.92 [sec] | 366.12 [sec] / 6.10 ± 0.57 [min] | 185 |
| AutoCF paper | - | - | - |
| AutoCF | 177507.10 [sec] / 2.05 [day] | 187.78 [sec] / 3.13 [min] | 378 |
| AutoCF no earlystopping | 184665.00 [sec] / 2.14 [day] | 221.48 [sec] / 3.69 [min] | 320 |

## 4.2. Bootstrap Latent Representations for Multi-modal Recommendation

In [91], published at World Wide Web Conference 2023 (WWW23), the authors present a Bootstrapped Multi-Modal Model (BM3), a self-supervised multi-modal recommendation model. Specifically, BM3 bootstraps latent contrastive views from the representations of users and items with a simple dropout augmentation. It then optimizes three multi-modal objectives to learn the representations of users and items. BM3 tries to alleviate both the need for contrasting with negative examples and the complex graph augmentation from an additional target network for contrastive view generation. In fact, it doesn't require augmentations from auxiliary graphs or negative samples. The model simplifies the current Self-Supervised Learning (SSL) framework by removing its target network, which can reduce half of the model parameters. To retain the similarity between different augmentations, BM3 incorporates a simple dropout mechanism to perturb the latent embeddings generated from the online network. Lastly, they designed a loss function that is specialized for multi-modal recommendation. It minimizes the reconstruction loss of the user-item interaction graph as well as aligns the learned features under both inter-modality and intra-modality perspectives.

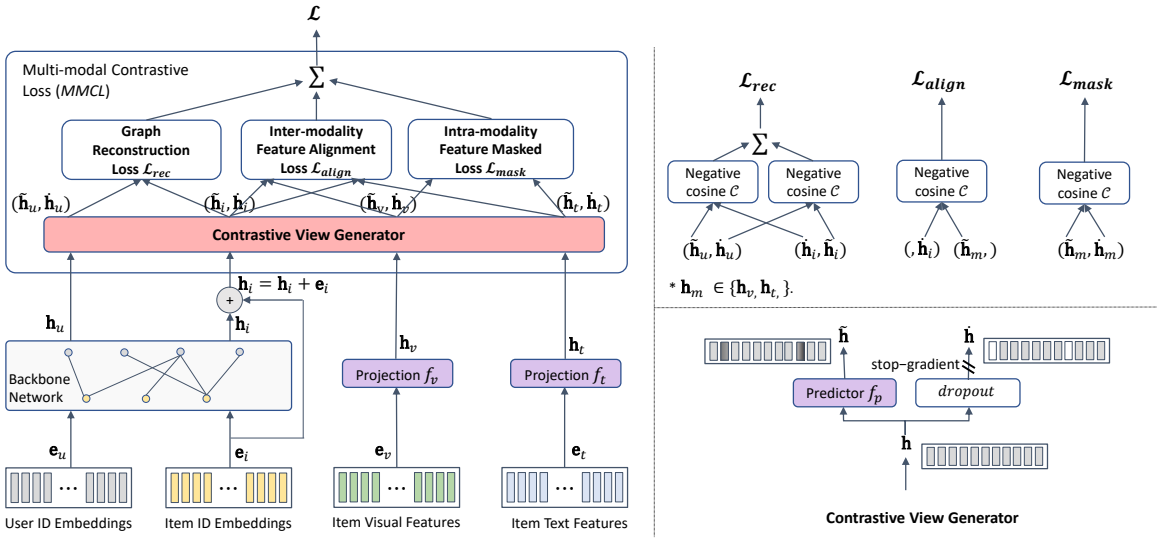


Figure 4.2: The structure overview of the proposed BM3 model. Projections $f_v$ and $f_t$, as well as predictor $f_p$, are all one layer of the Multi-Layer Perceptron (MLP). The parameters of predictor $f_p$ are shared in the Contrastive View Generator (*bottom left*) for ID embeddings and multi-modal latent representations.

To summarize their main contributions are:

1. A simple latent representation dropout mechanism instead of graph augmentation to generate the target view of a user or an item for contrastive learning without negative samples.

2. A Multi-Modal Contrastive Loss (MMCL) function that in order to train BM3 without negative samples jointly optimizes three objectives. In addition to minimizing the classic user-item interaction graph reconstruction loss, MMCL aligns the learned features between different modalities and reduces the dissimilarity between representations of different augmented views from a specific modality.

### 4.2.1. Datasets

The paper uses Amazon review dataset [2] [23] for experimental evaluation, their statistics are reported in Table 4.8. This dataset provides both product descriptions and images simultaneously, and it is publicly available and varies in size under different product categories. To ensure as many baselines can be evaluated on large-scale datasets, the authors choose three per-category datasets, Baby, Sports and Outdoors (denoted by Sports), and Electronics. We decided not to run the experiment on Amazon Electronics due to the high computational power required. In these datasets, all existing interactions are im-

[2]Datasets are available at http://jmcauley.ucsd.edu/data/amazon/links.html

plicit. The raw data of each dataset are pre-processed with a 5-core setting on both items and users, but it is not stated which criteria they used and how they performed the pre-processing.
The data sparsity is measured as the number of interactions divided by the product of the number of users and the number of items. The pre-processed datasets include both visual and textual modalities. This article was analyzed because it uses pre-trained embeddings for items and does not try to learn them during the recommendation process. Following [88], they use the 4,096-dimensional visual features that have been extracted and published in [43]. For the textual modality, they extract textual embeddings by concatenating the title, descriptions, categories, and brand of each item and utilize sentence-transformers [50] to obtain 384-dimensional sentence embeddings.

| Dataset | Interactions | Items | Users | Sparsity |
|---|---|---|---|---|
| Amazon Baby | 160792 | 7050 | 19445 | 9.988E-01 |
| Amazon Sports | 296337 | 18357 | 35598 | 9.995E-01 |

Table 4.8: BM3 Dataset characteristics

The characteristics (size and density) of the datasets we used are consistent with the ones reported in the original article.

### 4.2.2. Evaluation protocol

The paper followed the same evaluation setting of [70][88] with a random data splitting 8:1:1 on the interaction history of each user for training, validation and testing. The evaluation is performed reporting Recall@$K$ and NDCG@$K$ at cutoff 10 and 20 to evaluate the top-$K$ recommendation performance of different recommendation methods. Specifically, they use the all-ranking protocol instead of the negative-sampling protocol to compute the evaluation metrics for recommendation accuracy comparison. In the recommendation phase, all items that have not been interacted by the given user are regarded as candidate items. In order to keep the results presented in this report consistent, we used the dataset already preprocessed given in the original source code. Unfortunately, no information on how they are processed is given.

### 4.2.3. Baselines

The paper reports the following baseline as state-of-the-art recommendation methods, including general CF recommendation models and multi-modal recommendation models:

- **BPR** [52]: This is a matrix factorization model optimized by a pair-wise ranking

loss in a Bayesian way.

- **LightGCN** [25]: This is a simplified graph convolution network that only performs linear propagation and aggregation between neighbors. The hidden layer embeddings are averaged to calculate the final user and item embeddings for prediction.
- **BUIR** [33]: This self-supervised framework uses asymmetric network architecture to update its backbone network parameters. In BUIR, LightGCN is used as the backbone network. It is worth noting that BUIR does not rely on negative samples for learning.
- **VBPR** [22]: This model incorporates visual features for user preference learning with BPR loss. Following [70][88], the authors of the paper concatenate the multi-modal features of an item as its visual feature for user preference learning.
- **MMGCN** [74]: This method constructs a modal-specific graph to learn user preference on each modality leveraging GCN. The final user and item representations are generated by combining the learned representations from each modality.
- **GRCN** [75]: This method improves previous GCN-based models by refining the user-item bipartite graph with removal of false-positive edges. User and item representations are learned on the refined bipartite graph by performing information propagation and aggregation.
- **DualGNN** [70]: This method builds an additional user-user correlation graph from the user-item bipartite graph and uses it to fuse the user representation from its neighbors in the correlation graph.
- **LATTICE** [88]: This method mines the latent structure between items by learning an item-item graph from their multi-modal features. Graph convolutional operations are performed on both item-item graph and user-item interaction graph to learn user and item representations.

### 4.2.4. Hyperparameters

As other existing work [25][88], they fix the embedding size of both users and items to 64 for all models, initialize the embedding parameters with the Xavier method [20], and use Adam [28] as the optimizer with a learning rate of 0.001.
They tune the parameters of each model following their published papers. The proposed BM3 model is implemented by PyTorch [45]. They performed a grid search across all datasets to conform to their optimal settings. Specifically, the number of GCN layers is

tuned in {1, 2}. The dropout rate for embedding perturbation is chosen from {0.3, 0.5}, and the regularization coefficient is searched in {0.1, 0.01}. For convergence consideration, the early stopping and total epochs are fixed at 20 and 1000, respectively. Following [88], they use Recall@20 on the validation data as the training-stopping indicator. For this reason, we decided to optimize Recall@20 in our experiment. The optimal hyperparameters used for each dataset are mentioned in the public source code and are reported in Table 4.9. We used the same values in our experiment.

| Hyperparameter | Described in | Value | | |
|---|---|---|---|---|
| | | All datasets | Amazon Baby | Amazon Sports |
| Learning rate | Paper | 0.001 | - | - |
| Epochs | Paper | 1000 | - | - |
| Batch size | Source Code | 2048 | - | - |
| Embedding size | Paper | 64 | - | - |
| Regularization coefficient | Source code | - | 0.1 | 0.01 |
| Dropout rate | Source code | - | 0.5 | 0.5 |
| GCN layer | Source code | - | 1 | 1 |

Table 4.9: BM3 Hyperparameters values

### 4.2.5. Results and Discussion

The source code is publicly available on GitHub platform[3]. Tables 4.10 and 4.11 report the results contained in the original paper and the results obtained running the ported version of the algorithm when evaluated with the evaluator object of our framework.
In order to validate the results presented in the paper, we use the provided training-validation-test data split for all the datasets.
We were able to reproduce consistent results with the ones declared in the original work for both Amazon Baby and Amazon Sports datasets.
From the results is clearly visible that BM3 is largely outperformed by the simple and faster ItemKNN CF and RP3beta methods. In the Appendix A.0.2 we show full results for all metrics at different cutoffs.
Analyzing the beyond-accuracy metrics we can clearly see that in the case of the Amazon Baby dataset BM3 model without early stopping has greater performances on all the diversity metrics but this is the consequence of having much lower performance in terms of accuracy metrics. While for the Amazon Sports dataset which has better accuracy performances all the diversity metrics are lower than the baseline ones.
Analyzing the training time and recommendation time, taking the Amazon Baby dataset as a reference, we can state that the reproduced algorithm is faster than the Matrix

[3] https://github.com/enoche/BM3

Factorization method MF BPR while it is a lot slower than all the other baseline methods. We can further notice the good recommendation time of the BM3 algorithms which is faster than all the baselines. The time analysis table of Amazon Sport datasets can be found in Appendix A.0.2

Table 4.10: Experimental results for the BM3 method on the Amazon Baby dataset.

| | @ 10 | | @ 20 | |
|---|---|---|---|---|
| | REC | NDCG | REC | NDCG |
| TopPop | 0.0299 | 0.0162 | 0.0500 | 0.0213 |
| ItemKNN CF cosine | **0.0631** | **0.0357** | **0.0922** | **0.0431** |
| RP3beta | **0.0673** | **0.0387** | **0.0965** | **0.0462** |
| GraphFilter CF | 0.0560 | **0.0314** | 0.0854 | **0.0389** |
| MF BPR | 0.0325 | 0.0180 | 0.0537 | 0.0233 |
| PureSVD | 0.0374 | 0.0204 | 0.0599 | 0.0261 |
| BM3 paper | 0.0564 | 0.0301 | 0.0883 | 0.0383 |
| BM3 | 0.0570 | 0.0307 | 0.0888 | 0.0388 |
| BM3 no earlystopping | 0.0415 | 0.0225 | 0.0638 | 0.0283 |

Table 4.11: Experimental results for the BM3 method on the Amazon Sports dataset.

| | @ 10 | | @ 20 | |
|---|---|---|---|---|
| | REC | NDCG | REC | NDCG |
| TopPop | 0.0172 | 0.0098 | 0.0284 | 0.0126 |
| ItemKNN CF cosine | **0.0781** | **0.0454** | **0.1079** | **0.0530** |
| RP3beta | **0.0832** | **0.0486** | **0.1144** | **0.0565** |
| GraphFilter CF | 0.0685 | 0.0380 | 0.1006 | 0.0462 |
| MF BPR | 0.0194 | 0.0095 | 0.0309 | 0.0124 |
| PureSVD | 0.0408 | 0.0241 | 0.0596 | 0.0288 |
| BM3 paper | 0.0656 | 0.0355 | 0.0980 | 0.0438 |
| BM3 | 0.0672 | 0.0365 | 0.1027 | 0.0456 |
| BM3 no earlystopping | 0.0701 | 0.0388 | 0.1045 | 0.0477 |

Table 4.12: Experimental results for the BM3 method on the Amazon Baby dataset on beyond accuracy metrics.

| | @ 20 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0248 | 0.0311 | 0.0041 | 0.0033 | 0.0030 | 4.3918 |
| ItemKNN CF cosine | 0.0270 | 0.7671 | 0.4732 | 0.0603 | 0.0314 | 7.6930 |
| RP3beta | 0.0277 | 0.8369 | 0.6521 | 0.0713 | 0.0505 | 8.3318 |
| GraphFilter CF | 0.0292 | 0.8798 | 0.5672 | 0.0616 | 0.0798 | 8.9143 |
| MF BPR | 0.0257 | 0.1756 | 0.0077 | 0.0065 | 0.0035 | 4.8004 |
| PureSVD | 0.0268 | 0.7598 | 0.0433 | 0.0240 | 0.0124 | 6.8594 |
| BM3 paper | - | - | - | - | - | - |
| BM3 | 0.0268 | 0.7357 | 0.2298 | 0.0389 | 0.0175 | 7.1650 |
| BM3 no earlystopping | **0.0325** | **0.9805** | **0.8525** | **0.0828** | **0.2180** | **10.9101** |

Table 4.13: Experimental results for the BM3 method on the Amazon Sports dataset on beyond accuracy metrics.

| | @ 20 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0104 | 0.0195 | 0.0019 | 0.0013 | 0.0012 | 4.3710 |
| ItemKNN CF cosine | 0.0124 | 0.9306 | **0.7508** | **0.0736** | **0.0783** | 10.0214 |
| RP3beta | 0.0125 | 0.9435 | **0.8258** | **0.0805** | **0.0965** | 10.3367 |
| GraphFilter CF | **0.0131** | 0.9502 | **0.6344** | 0.0632 | **0.1002** | **10.5082** |
| MF BPR | 0.0118 | 0.7780 | 0.0106 | 0.0069 | 0.0049 | 6.8835 |
| PureSVD | 0.0121 | 0.9265 | 0.0723 | 0.0305 | 0.0167 | 8.6760 |
| BM3 paper | - | - | - | - | - | - |
| BM3 | 0.0122 | 0.9277 | 0.3091 | 0.0503 | 0.0285 | 9.1924 |
| BM3 no earlystopping | 0.0129 | **0.9651** | 0.5700 | 0.0652 | 0.0703 | 10.4326 |

Table 4.14: Computation time for the algorithms in the selected results for the BM3 method on the Amazon Baby dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.00 [sec] | 37.29 [sec] | 521 |
| ItemKNN CF cosine | 0.45 ± 0.10 [sec] | 39.99 ± 2.46 [sec] | 441 |
| RP3beta | 2.25 ± 0.70 [sec] | 39.53 ± 5.56 [sec] | 424 |
| GraphFilter CF | 53.36 ± 37.47 [sec] | 49.03 ± 7.95 [sec] | 503 |
| MF BPR | 246.55 [sec] / 4.11 ± 3.96 [min] | 28.25 ± 7.88 [sec] | 914 |
| PureSVD | 18.33 ± 29.82 [sec] | 78.77 [sec] / 1.31 ± 1.28 [min] | 340 |
| BM3 paper | - | - | - |
| BM3 | 181.60 [sec] / 3.03 [min] | 33.62 [sec] | 578 |
| BM3 no earlystopping | 452.63 [sec] / 7.54 [min] | 33.53 [sec] | 580 |

## 4.3. Multi-Modal Self-Supervised Learning for Recommendation

In [73], published at World Wide Web Conference 2023 (WWW23), the authors propose a new Multi-Modal Self-Supervised Learning (MMSSL) method, that unifies the generative and contrastive SSL for modality-aware augmentation. They designed a modality-aware interactive structure learning paradigm via adversarial perturbations for data augmentation to characterize the inter-dependency between the user-item collaborative view and item multi-modal-semantic view. They pursue a generic solution to jointly capture modality-specific collaborative effects and cross-modality interaction dependency, in recognizing multi-modal user preference. In particular, at the first stage of the SSL paradigm, they propose to train an adversarial relation learning network with the perturbed modality-aware user-item dependency weights. In the paper is reported that this scheme allows the method to distill the useful multi-modal information and encode them into the latent user (item) embeddings facing the limitation of label scarcity. To tackle the challenge of adversarial learning over a sparse user-item connection matrix, they integrate the Gumbel-based projection and Wasserstein adversarial generation to mitigate the distribution gap. In addition, to capture the effects that user's modality-aware interaction patterns would interweave with each other, a cross-modal contrastive learning approach is introduced to jointly preserve the inter-modal semantic commonality and user preference diversity.

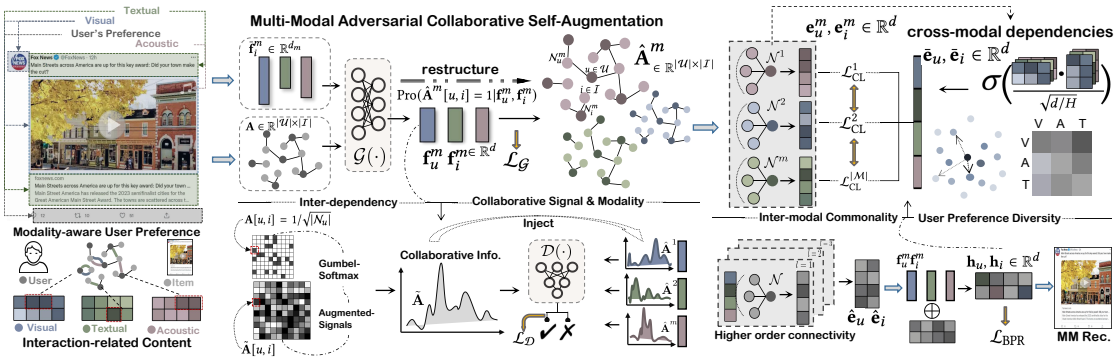


Figure 4.3: The model flow of MMSSL. Gumbel-based transformation is integrated with Wasserstein adversarial generation to mitigate distribution gap between the augmented user-item relation matrix $\hat{\mathbf{A}}^m$ generated via $\mathcal{G}(\cdot)$ and the original $\mathbf{A}$.

To summarize their contributions can be summarized as follows:

- They tackle the label scarcity issue of multimedia recommendation with dual-stage self-supervised learning for modality-aware data augmentation. By unifying the recommendation task and the augmented generative/contrastive multi-modal SSL signals.
- They propose a new recommender system MMSSL which integrates the generative modality-aware collaborative self-augmentation and the contrastive cross-modality dependency encoding.

### 4.3.1. Datasets

Experiments are conducted on four publicly available multi-modal recommendation datasets: Tiktok, Amazon Baby, Amazon Sports, and Allrecipes. All existing interactions are implicit. Their statistics are reported in Table 4.15.

**TikTok:** This data is collected from TikTok platform to log the viewed short-videos of users. The multi-modal characteristics are visual, acoustic, and title textual features of videos. The textual embeddings are encoded with Sentence-Bert [50].

**Amazon:** They adopt two benchmark datasets from Amazon with two item categories Amazon-Baby and Amazon-Sports [41]. In those datasets, textual feature embeddings are also generated via Sentence-Bert based on the extracted text from product title, description, brand and categorical information. The product images are used to generate 4096-d visual feature embeddings of items.

**Allrecipes:** This dataset comes from one of the largest food-oriented social network platform by including 52,821 recipes in 27 different categories. For each recipe, its

image and ingredients are considered as the visual and textual features. Following the setting in [19], 20 ingredients are sampled for each recipe.

| Dataset | Interactions | Items | Users | Sparsity |
|---|---|---|---|---|
| Amazon Baby | 160792 | 7050 | 19445 | 9.988E-01 |
| AllRecipes | 73494 | 10068 | 19805 | 9.996E-01 |
| Tiktok | 68722 | 6710 | 9308 | 9.989E-01 |

Table 4.15: MMSSL Dataset characteristics

We decided not to perform the experiment on the Amazon Sports dataset because we noticed that the experiment on this dataset was going to last more than 7 days on a Nvidia RTX 3090 GPU with 25GB of VRAM. We believe that three datasets are more than enough to successfully run our experiment. The characteristics, size, and density, of the datasets we used are consistent with the ones reported in the original article.

### 4.3.2. Evaluation protocol

The paper adopts three widely used metrics: Recall@K, Precision@K, and NDCG@K at cutoff 20. Following the settings in [75][77], the all-rank item evaluation strategy is used to measure the accuracy. The paper doesn't describe either the splitting method or how the validation split is done. Despite that, the original split is available in the source code[4].

### 4.3.3. Baselines

The paper reports the following baselines belonging to different categories: multi-modal recommender systems, popular GNN-based collaborative filtering models and recently proposed SSL-based recommendation solutions.

**1) GNN-based Collaborative Filtering Models**

- **NGCF** [71]: The method leverages graph convolutional network to inject high-order collaborative signals into representations.
- **LightGCN** [25]: By removing the redundant transformation and non-linear activation, LightGCN simplifies the message passing for graph neural network-based recommendation.

**2) SSL-based Recommendation Solutions**

- **SGL** [79]: This model improves the graph collaborative filtering with the incorpo-

[4] `https://github.com/HKUDS/MMSSL`

rated contrastive learning signals using different data augmentation operators, e.g., randomly node/edge dropout and random walk, to construct contrastive representation views.

- **NCL** [37]: In this approach, constrastive views are generated by identifying semantic and structural neighboring nodes with EM-based clustering, to generate the positive contrastive pairs.
- **HCCF** [80]: To supplement main recommendation objective with SSL task, HCCF leverages the hypergraph neural encoder to inject the global collaborative relations into the recommender.

**3) Multi-Modal Recommender Systems**

- **VBPR** [22]: This is a representative work to incorporate multimedia features into the matrix decomposition framework.
- **LightGCN-*M***: This baseline is generated by using SOTA GNN-based CF model LightGCN as backbone and incorporate multi-modal item features during the graph-based message passing.
- **MMGCN** [74]: It leverages graph convolutional network to propagate the modality-specific embedding and capture the modality-related user preference for micro-video recommendation.
- **GRCN** [75]: It is a structure-refined GCN multimedia recommender system, which yields the refined interactions to identify false-positive feedback and eliminate noisy with pruning edges.
- **LATTICE** [88]: It proposes to identify latent item-item relations with the generated item homogeneous graph. Connections will be added among items with similar modality features.
- **CLCRec** [76]: This model addresses the item cold-start issue by enhancing item embeddings with multi-modal features using mutual information-based contrastive learning.
- **MMGCL** [85]: It incorporates the graph contrastive learning into recommender through modality edge dropout and masking.
- **SLMRec** [65]: This method designs data augmentation on multi-modal content with two components, i.e., noise perturbation over features and multi-modal pattern uncovering augmentation.

### 4.3.4. Hyperparameters

MMSSL model is implemented with Pytorch. They adopt AdamW [40] and Adam [28] as the optimizer for generator and discriminator with the learning rate search range of $\{4.5e^{-4}, 5e^{-4}, 5.4e^{-3}, 5.6e^{-3}\}$ and $\{2.5e^{-4}, 3e^{-4}, 3.5e^{-4}\}$, respectively. The decay of $L_2$ regularization term is searched in $\{1.2e^{-2}, 1.4e^{-2}, 1.6e^{-2}\}$. The embedding dimensionality of MMSSL and the other compared baseline methods is set to 64. The number of propagation layers over the graph structure is tuned from {1, 2, 3, 4}. In the sensitivity analysis of the hyperparameters, they report that their method performs the best with 2 or 3 GNN layers. We decided to use 2 in our reproducible experiment because from their analysis it seemed to be the most promising choice. It is also used as a default value in the original implementation. Following similar settings of [48], the augmentation factors $\zeta$ and $\lambda_1$ in addressing the distribution gap in the multi-modal adversarial self-augmentation paradigm are selected from the range of [0, 10, 20, 50, 100, 150] and [0, 1, 10, 20, 50, 100, 150], respectively. In the paper is stated that the best performance is obtained with $\zeta = 100$ and $\lambda_1$=1. Analyzing the source code we find out that a value of $\lambda_1$=0.3 is used. To understand better this choice we first runned the algorithm on dataset Tiktok with $\lambda_1$=0.3 and then with $\lambda_1$=1 and we figured out that a value of 0.3 gives slightly better results, so we decided to use this value. The embedding dimensionality $d$ of their model is searched from $(2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9)$. Lastly, they tune the hyperparameter $\tau$ from $(0, 0.9)$ to control the agreement strength between the instances of positive pairs. They reported that from the results on TikTok dataset, the best accuracy is achieved with $\tau = 0.085$. The paper does not describe which metric is optimized. We decided to optimize NDCG@20 because is the metric used by the authors to generate more statistics besides the method accuracy. The criteria to select the number of epochs is not specified, they are set to 1000. The hyperparameters used for our reproducibility experiment are reported in Table 4.16.

| Hyperparameter | Described in | Value for all datasets |
|---|---|---|
| Learning rate | Source code | 0.00055 |
| Embedding Size | Paper | 64 |
| Self-Attention Heads | Source code | 4 |
| Batch size | Source Code | 1024 |
| Epochs | Source code | 1000 |
| Dropout Rate | Source code | 0.2 |
| $\lambda_1$ | Source code | 0.3 |
| $\tau$ | Paper | 0.085 |
| GNN layers | Paper | 2 |

Table 4.16: MMSSL Hyperparameters values

### 4.3.5. Results and Discussion

The source code is publicly available[5] on GitHub platform. Table 4.17, 4.18 and 4.19 report the results obtained running our baselines, the results contained in the original paper, and lastly the results obtained with the ported version of the algorithm when evaluated with the evaluator object of the framework. In order to validate the results presented in the paper, we use the provided training-validation-test data split.
From the results, we noticed that we are able to successfully reproduce the algorithm. To what concern Amazon Baby and Tiktok we can state that the method seems to produce slightly better results with respect to all baselines.
Looking at the diversity metrics we can further notice that this method has good abilities to leverage a wider number of items compared to the baselines for the Amazon Baby dataset. Instead looking at the AllRecipes results we notice a very unbalanced dataset, in fact, we can see how the non-personalized method like TopPop overperforms all the other methods. This is because in the dataset there are only a few items with most of the interactions. So trying to personalize the recommendation and use all the catalogs became a very hard task. Probably this is also not a good dataset to perform this type of evaluation.
Lastly, we need to highlight the different orders of magnitude in training time compared to the baseline. Baselines range in the order of seconds while the MMSSL method takes minutes in the early stopping run and hours in the NO early stopping run. In the Appendix A.0.3 full results, beyond accuracy metrics, and time tables are available.

Table 4.17: Experimental results for the MMSSL method on the Amazon Baby dataset.

| | @ 20 | | |
|---|---|---|---|
| | REC | PREC | NDCG |
| TopPop | 0.0464 | 0.0025 | 0.0194 |
| ItemKNN CF cosine | 0.0925 | 0.0049 | **0.0424** |
| RP3beta | 0.0886 | 0.0047 | 0.0409 |
| GraphFilter CF | 0.0929 | 0.0049 | 0.0401 |
| MF BPR | 0.0292 | 0.0015 | 0.0132 |
| PureSVD | 0.0652 | 0.0035 | 0.0274 |
| MMSSL paper | 0.0962 | 0.0051 | 0.0422 |
| MMSSL | **0.0970** | **0.0051** | 0.0423 |
| MMSSL no earlystopping | 0.0516 | 0.0027 | 0.0230 |

[5] https://github.com/HKUDS/MMSSL

Table 4.18: Experimental results for the MMSSL method on the AllRecipes dataset.

| | @ 20 | | |
|---|---|---|---|
| | REC | PREC | NDCG |
| TopPop | **0.0511** | **0.0026** | **0.0205** |
| ItemKNN CF cosine | 0.0306 | 0.0015 | 0.0123 |
| RP3beta | 0.0313 | 0.0016 | 0.0122 |
| GraphFilter CF | 0.0313 | 0.0016 | 0.0124 |
| MF BPR | 0.0249 | 0.0012 | 0.0095 |
| PureSVD | **0.0523** | **0.0026** | **0.0188** |
| MMSSL paper | 0.0367 | 0.0018 | 0.0135 |
| MMSSL | 0.0382 | 0.0019 | 0.0148 |
| MMSSL no earlystopping | 0.0175 | 0.0009 | 0.0066 |

Table 4.19: Experimental results for the MMSSL method on the Tiktok dataset.

| | @ 20 | | |
|---|---|---|---|
| | REC | PREC | NDCG |
| TopPop | 0.0798 | 0.0040 | 0.0416 |
| ItemKNN CF cosine | 0.1117 | 0.0056 | 0.0523 |
| RP3beta | 0.1207 | 0.0060 | **0.0563** |
| GraphFilter CF | 0.1132 | 0.0057 | 0.0495 |
| MF BPR | 0.0502 | 0.0025 | 0.0213 |
| PureSVD | 0.0550 | 0.0027 | 0.0279 |
| MMSSL paper | 0.0921 | 0.0046 | 0.0392 |
| MMSSL | **0.1215** | **0.0061** | 0.0533 |
| MMSSL no earlystopping | 0.0850 | 0.0042 | 0.0368 |

Table 4.20: Experimental results for the MMSSL method on the Amazon Baby dataset on beyond accuracy metrics.

| | @ 20 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0248 | 0.0314 | 0.0041 | 0.0034 | 0.0030 | 4.3923 |
| ItemKNN CF cosine | 0.0271 | 0.7800 | 0.5048 | 0.0621 | 0.0360 | 7.8536 |
| RP3beta | 0.0269 | 0.7568 | 0.4596 | 0.0570 | 0.0304 | 7.6444 |
| GraphFilter CF | 0.0283 | 0.7963 | 0.7881 | 0.0584 | 0.0643 | 8.1330 |
| MF BPR | 0.0273 | 0.5134 | 0.0072 | 0.0061 | 0.0058 | 5.3827 |
| PureSVD | 0.0266 | 0.7310 | 0.0374 | 0.0224 | 0.0110 | 6.6895 |
| MMSSL paper | - | - | - | - | - | - |
| MMSSL | **0.0289** | **0.9303** | 0.3721 | **0.0706** | 0.0537 | **8.9288** |
| MMSSL no earlystopping | **0.0325** | **0.9866** | **0.9169** | **0.0848** | **0.2577** | **11.2229** |

Table 4.21: Computation time for the algorithms in the selected results for the MMSSL method on the Amazon Baby dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.00 [sec] | 37.28 [sec] | 522 |
| ItemKNN CF cosine | 0.29 ± 0.05 [sec] | 40.85 ± 3.76 [sec] | 445 |
| RP3beta | 1.72 ± 0.44 [sec] | 40.48 ± 2.80 [sec] | 445 |
| GraphFilter CF | 24.00 ± 8.52 [sec] | 46.33 ± 2.63 [sec] | 405 |
| MF BPR | 407.18 [sec] / 6.79 ± 6.90 [min] | 37.85 ± 1.81 [sec] | 563 |
| PureSVD | 1.31 ± 0.91 [sec] | 45.54 ± 2.95 [sec] | 392 |
| MMSSL paper | - | - | - |
| MMSSL | 1665.64 [sec] / 27.76 [min] | 36.98 [sec] | 526 |
| MMSSL no earlystopping | 29965.38 [sec] / 8.32 [hour] | 35.39 [sec] | 549 |

## 4.4. Fast Variational AutoEncoder with Inverted Multi-Index for Collaborative Filtering

In [7], published at World Wide Web Conference 2022 (WWW22), the authors propose a fast Variational AutoEncoder (VAE) with Inverted Multi-Index for Collaborative Fil-

tering. The bottleneck of recent VAEs algorithms lies in the softmax computation over all items, such that it takes linear costs in the number of items to compute the loss and gradient for optimization. In recent years, Variational AutoEncoder [29] has been extended as a representative nonlinear method (Mult-VAE) for recommendation [35], and received much attention among the recommender system community [49][57][62]. Mult-VAE encodes each user's observed data with a Gaussian-distributed latent factor and decodes it to a probability distribution over all items, which is assumed a softmax of the inner-product-based logits. Mult-VAE then exploits multinomial likelihood as the objective function for optimization, which has been proved to be more tailored for implicit feedback than the Gaussian and logistic likelihood. However, the bottleneck of Mult-VAE lies in the log-partition function over the logits of all items in the multinomial likelihood. The time to compute the loss and gradient in each training step grows linearly with the number of items. When there are an extremely large number of items, the training of Mult-VAE is time-consuming, making it impractical in real recommendation scenarios. To this end, based on the popular MIPs index – inverted multi-index [3], they propose a series of proposal distributions, from which items can be efficiently yet independently sampled, to approximate the softmax distribution. The basic idea is to decompose item sampling into multiple stages.

Their main contributions can be summarized as follows:

- They study high-quality approximated softmax distributions, by decomposing the softmax probability based on the inverted multi-index.
- They design a sampling process for these approximate softmax distributions, from which items can be independently sampled in sublinear or constant time. These samplers are applied for developing the fast Variational AutoEncoder.
- They demonstrate that FastVAE performs at least as well as VAE for recommendation. Moreover, in the original paper, it is claimed that the proposed samplers are highly accurate compared to existing sampling methods, and perform sampling with high efficiency.

### 4.4.1. Datasets

The evaluation is performed on four public datasets described and processed as follows, their statistics are reported in Table 4.22. All existing interactions are implicit and assigned a value of either 0 or 1.

**MovieLens10M:** shorted as ML10M, is a classic movie rating dataset, whose ratings

range from 0.5 to 5. They convert them into 0/1 indicating whether the user has rated the movie.

**Gowalla:** includes users' check-ins at locations in a location-based social network and is much sparser than the MovieLens dataset.

**Netflix:** is another famous movie rating dataset but with many more users taken from the Netflix platform.

**Amazon:** is a subset of customers' ratings for Amazon books, where the rating scores are integers from 1 to 5, and books with scores higher than 4 are considered positive.

For all the datasets, they filter out users and items with less than 10 interactions. The characteristics, size and density, of the datasets we are using for the experiments are consistent with the ones reported in the original article. Only the MovieLens10M dataset with implicit records was available in the source code but not preprocessed as they mentioned so we managed to remove all users and items with less than 10 interactions. Then we managed to download and preprocess the Gowalla dataset, in the same way, as reported in the paper. We decided to run the experiment only on Gowalla and ML10M because they stated that these two datasets are the ones in which their model overperforms the baselines they used while Amazon and Netflix didn't overperform the baselines.

| Dataset | Interactions | Items | Users | Sparsity |
|---|---|---|---|---|
| MovieLens10M | 2001164 | 5942 | 47292 | 9.929E-01 |
| Gowalla | 1027464 | 40988 | 29858 | 9.992E-01 |

Table 4.22: FastVAE Dataset characteristics

### 4.4.2. Evaluation protocol

Two standard metrics are utilized for evaluating the quality of recommendation, Recall@K and NDCG@K at cutoff 50. For each user, they randomly sample 80% of interacted items to construct the history vector and fit the models to the training items. They do not use early stopping so there is no validation data. In the paper is stated that they performed 5 times cross-validation, but no other detail is given. All algorithms are fine-tuned based on NDCG@50. We therefore utilize NDCG@50 as a metric to optimize in our reproducibility experiment. The original train-test split of the data is not available on the Github repository, but in the source code is given the function to perform the split. We decided to follow the same splitting strategy but the training set was then split again with a ratio of 0.8 to create a validation and perform early stopping. So our dataset is

partitioned as follows: the training set is 64%, the validation set is 16% and the test is 20% of the entire dataset.

### 4.4.3. Baselines

The paper reports the following baselines belonging to collaborative filtering models:

- **WRMF** [27][44], weighted regularized matrix factorization, is a famous collaborative filtering method for implicit feedback. It sets a prior on uninteracted items associated with the confidence level of being negative. It learns parameters by alternating least square method in the case of square loss.
- **BPR** [52], Bayesian personalized ranking for implicit feedback, utilizes the pairwise logit ranking loss between positive and negative samples. For each pair of interacted user and item, BPR randomly samples several uninteracted items of the user for training and applies stochastic gradient descent for optimization.
- **WARP-MF** [78] uses the weighted approximate-rank pair-wise loss function for collaborative filtering. Given a positive item, it uniformly samples negative items until the rating of the sampled item is higher. The rank is estimated based on the sampling trials. They use the implementation in the lightFM[6].
- **AOBPR** [51] improves the BPR with adaptive sampling method. They use the version implemented in LibRec[7].
- **DNS** [89] dynamically chooses items according to the predicted ranking list for the top-k recommendation. Specifically, the dynamic sampler first draws samples uniformly from the item set and the item with the maximum rating is selected.
- **PRIS** [34] utilizes the importance sampling to the pairwise ranking loss for personalized ranking and assigns the sampling weight to the sampled items. We adopt the joint model implemented in the open resource code[8].
- **Self-Adversarial (SA)** [64], a self-supervised method for negative sampling, is recently proposed for the recommendation. It utilizes uniform sampling and assigns the sampling weight for the negative item depending on the current model.
- **Mult-VAE** [35], variational autoencoders for collaborative filtering. It learns the user representation by aiming at maximizing the likelihood of user click history.

[6]https://github.com/lyst/lightfm
[7]https://github.com/guoguibing/librec
[8]https://github.com/DefuLian/PRIS

In addition to these recommendation algorithms, they conduct experiments with the following samplers.

- **Uniform** sampler is a common sampling strategy that randomly draws negatives from the set of items for optimization, widely used for sampled softmax.
- **Popularity** sampler is correlated with the popularity of items, where the items with higher popularity have a greater probability of being sampled. The popularity is computed as $\log(f_i + 1)$ where $f_i$ is the occurring frequency of item $i$. They normalize the popularity of all items for sampling.
- **Kernel** based sampler [5] is a recent method for adaptively sampled softmax, which lowers the bias by the non-negative quadratic kernel. Furthermore, the kernel-based sampler is implemented with divide and conquer depending on the tree structure.

### 4.4.4. Hyperparameters

They developed the proposed algorithms FastVAE with Pytorch. They utilize the Adam algorithm with a weight decay of 0.01 for optimization. They implement the variational autoencoder with one hidden layer and the generative module would be $[M \to 200 \to 32]$. The active function between layers is ReLu by default. Dropout is set with a probability of 0.5. The batch size is set to 256 for all the datasets. The learning rate is tuned over {0.1,0.01,0.001,0.0001}. They train the models within 200 epochs, even if the criteria to select the number of epochs is not specified.
For the FastVAE, the number of samples is set to 200 for the MovieLens10M and Netflix dataset, 1000 for the Gowalla dataset, and 2000 for the Amazon dataset.
Regarding the popularity-based strategy, they follow the same popularity function $\log(f_i+1)$. For most of the baselines, hyperparameters are specified with their search space. The hyperparameters used for our reproducibility experiment are reported in Table 4.23.

| Hyperparameter | Described in | Value | | |
|---|---|---|---|---|
| | | All datasets | MovieLens10M | Gowalla |
| Regularization rate | Paper | 0.01 | - | |
| Learning rate | Source code | 0.001 | - | |
| Batch size | Paper | 256 | - | |
| Epochs | Paper | 200 | - | |
| Dropout | Paper | 0.5 | - | |
| Weight Decay | Paper | 1 | - | |
| Number of samples | Paper | - | 200 | 1000 |

Table 4.23: FastVAE Hyperparameters values

### 4.4.5. Results and Discussion

The source code is publicly available[9] on the GitHub platform. Table 4.24 and 4.25 shows the values obtained running our baselines, the values contained in the original paper, and lastly the results obtained with the ported version of the algorithm when evaluated with the evaluator object of the framework. In order to try to validate the performance, we preprocessed the publicly available dataset as reported in the paper. We ran the experiment for 200 epochs as reported in the original work.

Contrary to the experiment done previously we were not able to reproduce this algorithm in fact the results we obtained in terms of NDCG@50 in our framework are lower than 8% in the case of MovieLens10M and around 17% in the case of Gowalla. Even if the performance gap between our experiment and the original paper is really important it could be due to a different split of the dataset. Since the results with 200 epochs were not achieved we tried to slightly increase the number of epochs to 500 but it did not change much. In fact, only the non-early stopping performance increased by a very insignificant amount. The paper explains that they performed a 5 times cross-validation without giving any other details. These unclear and undocumented steps are often the cause of incomprehension and non-transparent work which has been revealed to be non-reproducible.

By comparing the values obtained from the baselines with the values stated in the paper, for the Gowalla dataset, FastVAE would not have been competitive in any case. Whereas if FastVAE had been able to achieve the values mentioned in the paper for the MovieLens10M dataset it would have been very competitive against the baselines.

Looking at the beyond accuracy metrics we can notice that MovieLens10M, which has slightly lower recommendation accuracy compared to RP3Beta and GraphFilter CF, has good abilities to diversify the recommendation leveraging in a more complete way the catalog of items.

In terms of training time FastVAE is slower than all the baselines except MF BPR while it is slightly faster in terms of recommendation time. In Appendix A.0.4 full results for both datasets are available, while for the Gowalla dataset, there are also the beyond accuracy metrics, and the time tables.

[9] `https://github.com/HERECJ/FastVae_Gpu`

Table 4.24: Experimental results for the FastVAE method on the MovieLens10M dataset.

| | @ 50 | |
|---|---|---|
| | NDCG | REC |
| TopPop | 0.1636 | 0.2716 |
| ItemKNN CF cosine | 0.3018 | 0.4636 |
| RP3beta | **0.3138** | 0.4866 |
| GraphFilter CF | **0.3288** | **0.4992** |
| MF BPR | 0.2826 | 0.4571 |
| PureSVD | 0.2643 | 0.4113 |
| FastVAE paper | **0.3275** | **0.5078** |
| FastVAE | 0.2953 | 0.4815 |
| FastVAE no earlystopping | 0.3030 | 0.4882 |

Table 4.25: Experimental results for the FastVAE method on the Gowalla dataset.

| | @ 50 | |
|---|---|---|
| | NDCG | REC |
| TopPop | 0.0275 | 0.0493 |
| ItemKNN CF cosine | **0.1974** | **0.3172** |
| RP3beta | **0.2087** | **0.3382** |
| GraphFilter CF | **0.2038** | **0.3355** |
| MF BPR | 0.1422 | 0.2402 |
| PureSVD | 0.1413 | 0.2289 |
| FastVAE paper | **0.1797** | **0.2971** |
| FastVAE | 0.1433 | 0.2476 |
| FastVAE no earlystopping | 0.1487 | 0.2566 |

Table 4.26: Experimental results for the FastVAE method on the MovieLens10M dataset on beyond accuracy metrics.

| | @ 50 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0685 | 0.1861 | 0.0247 | 0.0234 | 0.0098 | 6.0305 |
| ItemKNN CF cosine | 0.0747 | 0.6966 | 0.5618 | 0.2686 | 0.0366 | 8.0774 |
| RP3beta | 0.0770 | 0.7677 | **0.7109** | 0.3263 | 0.0505 | 8.5218 |
| GraphFilter CF | 0.0768 | 0.7809 | 0.4377 | 0.2558 | 0.0473 | 8.5167 |
| MF BPR | 0.0786 | 0.8242 | 0.4778 | 0.2787 | 0.0642 | 8.9340 |
| PureSVD | 0.0749 | 0.7512 | 0.0799 | 0.0739 | 0.0311 | 7.9223 |
| FastVAE paper | - | - | - | - | - | - |
| FastVAE | **0.0804** | **0.8541** | 0.5239 | **0.3775** | **0.0796** | **9.2400** |
| FastVAE no earlystopping | **0.0807** | **0.8569** | 0.5771 | **0.3990** | **0.0848** | **9.3128** |

Table 4.27: Computation time for the algorithms in the selected results for the FastVAE method on the MovieLens10M dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.02 [sec] | 139.79 [sec] / 2.33 [min] | 338 |
| ItemKNN CF cosine | 1.02 ± 0.07 [sec] | 153.18 [sec] / 2.55 ± 0.08 [min] | 318 |
| RP3beta | 7.32 ± 3.06 [sec] | 153.56 [sec] / 2.56 ± 0.09 [min] | 319 |
| GraphFilter CF | 22.10 ± 8.38 [sec] | 152.13 [sec] / 2.54 ± 0.35 [min] | 392 |
| MF BPR | 2422.56 [sec] / 40.38 ± 36.56 [min] | 90.65 [sec] / 1.51 ± 0.04 [min] | 514 |
| PureSVD | 2.75 ± 2.19 [sec] | 145.14 [sec] / 2.42 ± 0.03 [min] | 324 |
| FastVAE paper | - | - | - |
| FastVAE | 909.32 [sec] / 15.16 [min] | 115.41 [sec] / 1.92 [min] | 410 |
| FastVAE no earlystopping | 1060.64 [sec] / 17.68 [min] | 112.39 [sec] / 1.87 [min] | 421 |

## 4.5. Discussion of the Reproducibility Experiments

In this section, we are going to provide a discussion of the results we obtained with our analysis.

### 4.5.1. Reproducibility of the Papers

Applying our distinct methodological criteria, we discovered that 3 out of 9 relevant papers could be reproduced using the provided code and publicly accessible datasets. However, it is noteworthy that in half of the unreproduced papers, was not possible to conduct the experiment due to the absence of source code.

From a technical standpoint, a scientific paper is expected to encompass all essential details for the replication of the research. Despite this expectation, we encountered a notable number of publications where some necessary artifacts were missing. In our study, we reached out to authors of papers lacking publicly available code and datasets. While some authors were responsive and shared their code, in certain instances, our request was in vain, receiving no answer.

The omission of crucial technical specifics in some papers resulted in significantly divergent outcomes from the reported ones. Nevertheless, it is generally in the best interest of the scientific community and researchers themselves to facilitate the reproduction, validation, or refutation of their results, recognizing the significance of their contributions to scientific research.

Even when all the necessary information was available, it was never easy to reproduce the algorithm and work with other people's source code. It was almost always insufficiently commented and often disorganized in structure and key passages. We found that on more than one occasion hyperparameters were hardcoded into the code, this is a practice that needs to be avoided while in some cases hyperparameter values other than those highlighted as best or stated as being used to conduct their experiment were used.

Also reproducing the environment needed for each algorithm posed some challenges, despite relying on public libraries and freely available software tools the intricate relationships between software requirements added some difficulties to our reproducibility work. In some cases, we identified errors in the code attributed to missing imports of libraries or modifications in the employed library preventing us from executing the code, so we had to properly fix those errors to guarantee the successful running of the experiment.

### 4.5.2. Competitiveness with Baselines

Our study exposes that none of the neural approaches considered emerges as a competitive and robust algorithm demonstrating strong superior performance in the top-k recommendation task. Our findings demonstrate that, more often than not, even relatively straightforward methods such as KNN or Graph-based techniques yield performance levels comparable to or even surpass the latest neural methods.

Several factors contribute to this observation. We have identified various methodological issues, and notably, the selection of baselines and the frequent absence of proper optimization of these baselines emerge as significant issues affecting the stated results.
The outcomes of the experiments appear to affirm that, within the scope of the papers analyzed in this study (albeit a limited subset compared to the overall publications), there has been irrelevant substantial progress in recent years. When replicating the scenarios outlined in the original papers, we observed that in a minority of these instances, the newly proposed methods outperformed the simple baselines. Only in one case did we achieve commendable results comparable to the baselines, although this was not consistent across all tested datasets. As pointed out in [63] an open issue is to understand whether the datasets we have are truly representative of reality or not. If there are industries such as Netflix that use neural methods massively but we can never find an advantage with public datasets it means that we have usage scenarios that the research community struggles to work on because of the lack of appropriate data.

### 4.5.3. Evaluation Methodology Issues

In the examination of each paper, we documented several methodological issues that may contribute to elucidating the observed limited competitiveness of the neural methods under consideration in this study:

- In certain instances, authors do not give any theoretical or practical rationale for their decisions or elucidate why their models yielded the showcased results. Frequently, crucial assumptions were left unjustified or unmentioned, with no supporting evidence of their choices.
- Rather than being benchmarked against straightforward yet well-known baselines, new algorithms are often compared against other emerging methods as baselines. This approach tends to present the new neural methods as consistently outperforming in all scenarios. However, empirical evidence suggests that algorithm performance rankings hinge on various factors, encompassing dataset characteristics, evaluation procedures, and the metrics employed.
- We've observed a lack of comprehensive documentation or, in some cases, an absence of any description regarding the data split and preprocessing procedures.
- The criteria employed to determine the number of training epochs is typically not specified.
- There is a lack of standardization in the methodology employed for evaluation.

Across various approaches, disparities exist in several aspects, including datasets, preprocessing techniques, dataset splits, evaluation metrics, and cutoffs. Typically, there is no exhaustive explanation for the choices made in these aspects. This lack of standardization complicates the comparison of obtained results and raises doubts about the validity of claims that a new algorithm outperforms the state of the art, considering that such claims may be valid only in the specific conditions under which the evaluation was conducted.

### 4.5.4. Scalability

Given the substantial computational cost associated with a recommended system algorithm in terms of training time and scalability over extensive datasets in a real production environment, it becomes crucial to consider this aspect in new publications. When the system is implemented and employed in real scenarios involving thousands of users, the time needed for training the model and providing recommendations can significantly influence both costs and the quality of results. Despite this, only short comments are usually provided concerning this aspect.

Taking into consideration the result of our experiments we can easily notice that the time performance of all the neural methods is largely worse compared to the baselines despite no improvement in the recommendation accuracy is made.

# 5 | Conclusions

Over the past few years, an increasing number of papers have expressed strong interest in the validity of results claimed by authors presenting new methods for recommender systems [16][17][83].

The aim of this thesis was to enhance understanding of these concerns and to evaluate the advancements made in recent years by neural methods in the context of recommender systems, with a specific focus on top-K recommendations.

To attain this objective, an in-depth examination of papers presented at leading international conferences was carried out. The goal was to assess the replicability of results by implementing the proposed algorithms within a standard framework. This facilitated a comparison of outcomes with those generated by suitably tuned non-neural baseline algorithms, functioning in the same environment and under similar conditions.

Chapter 4 of this thesis, titled "Analysis and Results of Reproducible Papers," offers an extensive analysis of the outcomes produced for each examined paper, accompanied by a discussion of the various aspects under consideration.

Our work indicates that, despite the large number of new neural methods published in recent years, progress in this field appears to be constrained. This emerges particularly from the analysis of the papers discussed in this study. Applying our selection criteria, we discovered that only 3 out of 9 papers could be reproduced. Notably, in half of the not reproducible papers, the absence of provided source code was the main contributing factor.

Our examination reveals that, frequently, even relatively straightforward methods achieve performance levels comparable to or surpassing those of the latest intricate neural approaches. Several factors contribute to this observed reality. We outlined some methodological challenges, with the non-accurate selection of baselines and the often absence of proper optimization of these baselines being key issues hindering progress in this field. Additionally, we noted technical errors in the code and in the way, the evaluation is made that impacted the results presented by the papers.

During our reproducibility study, we found that many of the problems we highlighted had already been pointed out in previous reproducibility works.

Moreover, when examining methodological concerns in a recently proposed method, there is a potential risk that if authors claim that their work markedly outperforms baselines, that were not carefully chosen and tuned for each specific dataset, such an algorithm might be perceived as a strong baseline in future research works. Nonetheless, if the asserted results are attained under weak evaluation conditions, the widespread acceptance of this new baseline can result in poor evaluations for future algorithms. This, in turn, has the potential to distort the progress of research in recommendation systems. It's worth noting that these challenges are not unique to the domain of recommender systems or specific to neural approaches.

Furthermore, a common trend observed in nearly all screened papers involves evaluating the proposed algorithm's performance primarily using accuracy metrics, assuming that an enhancement in accuracy translates to greater user satisfaction. The pervasive focus on achieving the highest accuracy performance value dominates the literature under consideration. However, doubts regarding this assumption have been raised in several research papers [14] [56] [4], suggesting that greater accuracy does not necessarily translate to improved perceived recommendations for users. Instead, greater emphasis should be placed on other aspects, such as "beyond accuracy metrics", response time, and investigation about the reason why some items are recommended instead of others. It's worth noting that some of these characteristics cannot be adequately assessed through offline experiments.

Additionally, technical considerations such as scalability, training time, recommendation time, complexity, and robustness represent crucial aspects. These factors can significantly impact development and maintenance costs, as well as, user satisfaction. Despite their importance, only a limited number of papers presented at leading international conferences deal with these issues.

To address these challenges, the recommender systems research community should adopt a more systematic approach to evaluating progress when introducing new technologies. Enhancing the reproducibility of published articles stands out as a potential strategy to alleviate some of the identified issues.

Additionally, greater emphasis should be placed on refining the experimental evaluation process. Conference admission criteria should reflect this consideration, demanding a transparent explanation of the experimental protocol employed. Preference should be given to papers that present results across multiple scenarios and varied conditions, including those reporting negative outcomes. Alternatively, papers should clearly define the scenarios for which the algorithm was designed, justifying the choice of dataset, metrics, and all the relevant choices made.

In summary, this thesis does not assert that neural methods are unsuitable for applica-

tion in recommender systems. Instead, it emphasizes the necessity of comprehending the motivations and reasoning that can genuinely result in dependable enhancements in this domain. It underscores the importance of ensuring that practical results are consistently reproducible and endorsed by theoretical rationale.

# Bibliography

[1] G. Adomavicius and Y. Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Trans. Knowl. Data Eng.*, 24(5):896–911, 2012. doi: 10.1109/TKDE.2011.15. URL https://doi.org/10.1109/TKDE.2011.15.

[2] C. C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016. ISBN 978-3-319-29657-9. doi: 10.1007/978-3-319-29659-3. URL https://doi.org/10.1007/978-3-319-29659-3.

[3] A. Babenko and V. S. Lempitsky. The inverted multi-index. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(6):1247–1260, 2015. doi: 10.1109/TPAMI.2014.2361319. URL https://doi.org/10.1109/TPAMI.2014.2361319.

[4] J. Beel and S. Langer. A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In S. Kapidakis, C. Mazurek, and M. Werla, editors, *Research and Advanced Technology for Digital Libraries - 19th International Conference on Theory and Practice of Digital Libraries, TPDL 2015, Poznań, Poland, September 14-18, 2015. Proceedings*, volume 9316 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 2015. doi: 10.1007/978-3-319-24592-8_12. URL https://doi.org/10.1007/978-3-319-24592-8_12.

[5] G. Blanc and S. Rendle. Adaptive sampled softmax with kernel based sampling. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 589–598. PMLR, 2018. URL http://proceedings.mlr.press/v80/blanc18a.html.

[6] R. D. Burke. Hybrid recommender systems: Survey and experiments. *User Model. User Adapt. Interact.*, 12(4):331–370, 2002. doi: 10.1023/A:1021240730564. URL https://doi.org/10.1023/A:1021240730564.

[7] J. Chen, D. Lian, B. Jin, X. Huang, K. Zheng, and E. Chen. Fast variational autoencoder with inverted multi-index for collaborative filtering. In F. Laforest,

R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 1944–1954. ACM, 2022. doi: 10.1145/3485447.3512068. URL https://doi.org/10.1145/3485447.3512068.

[8] L. Chen, L. Wu, R. Hong, K. Zhang, and M. Wang. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 27–34. AAAI Press, 2020. doi: 10.1609/aaai.v34i01.5330. URL https://doi.org/10.1609/aaai.v34i01.5330.

[9] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In A. Karatzoglou, B. Hidasi, D. Tikk, O. S. Shalom, H. Roitman, B. Shapira, and L. Rokach, editors, *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016*, pages 7–10. ACM, 2016. doi: 10.1145/2988450.2988454. URL https://doi.org/10.1145/2988450.2988454.

[10] Z. Cheng, S. Han, F. Liu, L. Zhu, Z. Gao, and Y. Peng. Multi-behavior recommendation with cascading graph convolution networks. In Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, and G. Houben, editors, *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 1181–1189. ACM, 2023. doi: 10.1145/3543507.3583439. URL https://doi.org/10.1145/3543507.3583439.

[11] C. Cooper, S. Lee, T. Radzik, and Y. Siantos. Random walks in recommender systems: exact computation and simulations. In C. Chung, A. Z. Broder, K. Shim, and T. Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 811–816. ACM, 2014. doi: 10.1145/2567948.2579244. URL https://doi.org/10.1145/2567948.2579244.

[12] P. Cremonesi. *Lecture notes of the academic year course on "recommender systems"*. Politecnico di Milano, A.A. 2022/2023.

[13] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms

on top-n recommendation tasks. In X. Amatriain, M. Torrens, P. Resnick, and M. Zanker, editors, *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010*, pages 39–46. ACM, 2010. doi: 10.1145/1864708.1864721. URL https://doi.org/10.1145/1864708.1864721.

[14] P. Cremonesi, F. Garzotto, and R. Turrin. Investigating the persuasion potential of recommender systems from a quality perspective: An empirical study. *ACM Trans. Interact. Intell. Syst.*, 2(2):11:1–11:41, 2012. doi: 10.1145/2209310.2209314. URL https://doi.org/10.1145/2209310.2209314.

[15] M. F. Dacrema. Demonstrating the equivalence of list based and aggregate metrics to measure the diversity of recommendations (student abstract). In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 15779–15780. AAAI Press, 2021. doi: 10.1609/AAAI.V35I18.17886. URL https://doi.org/10.1609/aaai.v35i18.17886.

[16] M. F. Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In T. Bogers, A. Said, P. Brusilovsky, and D. Tikk, editors, *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*, pages 101–109. ACM, 2019. doi: 10.1145/3298689.3347058. URL https://doi.org/10.1145/3298689.3347058.

[17] M. F. Dacrema, S. Boglio, P. Cremonesi, and D. Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Trans. Inf. Syst.*, 39(2):20:1–20:49, 2021. doi: 10.1145/3434185. URL https://doi.org/10.1145/3434185.

[18] S. Debnath, N. Ganguly, and P. Mitra. Feature weighting in content based recommendation system using social network analysis. In J. Huai, R. Chen, H. Hon, Y. Liu, W. Ma, A. Tomkins, and X. Zhang, editors, *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 1041–1042. ACM, 2008. doi: 10.1145/1367497.1367646. URL https://doi.org/10.1145/1367497.1367646.

[19] X. Gao, F. Feng, X. He, H. Huang, X. Guan, C. Feng, Z. Ming, and T. Chua. Hierarchical attention network for visually-aware food recommendation. *IEEE Trans.*

*Multim.*, 22(6):1647–1659, 2020. doi: 10.1109/TMM.2019.2945180. URL https://doi.org/10.1109/TMM.2019.2945180.

[20] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and D. M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010. URL http://proceedings.mlr.press/v9/glorot10a.html.

[21] S. Gu, X. Wang, C. Shi, and D. Xiao. Self-supervised graph neural networks for multi-behavior recommendation. In L. D. Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 2052–2058. ijcai.org, 2022. doi: 10.24963/IJCAI.2022/285. URL https://doi.org/10.24963/ijcai.2022/285.

[22] R. He and J. J. McAuley. VBPR: visual bayesian personalized ranking from implicit feedback. In D. Schuurmans and M. P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 144–150. AAAI Press, 2016. doi: 10.1609/aaai.v30i1.9973. URL https://doi.org/10.1609/aaai.v30i1.9973.

[23] R. He and J. J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In J. Bourdeau, J. Hendler, R. Nkambou, I. Horrocks, and B. Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 507–517. ACM, 2016. doi: 10.1145/2872427.2883037. URL https://doi.org/10.1145/2872427.2883037.

[24] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua. Neural collaborative filtering. In R. Barrett, R. Cummings, E. Agichtein, and E. Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 173–182. ACM, 2017. doi: 10.1145/3038912.3052569. URL https://doi.org/10.1145/3038912.3052569.

[25] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In J. X. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, and Y. Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 639–648. ACM,

2020. doi: 10.1145/3397271.3401063. URL https://doi.org/10.1145/3397271.3401063.

[26] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004. doi: 10.1145/963770.963772. URL https://doi.org/10.1145/963770.963772.

[27] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 263–272. IEEE Computer Society, 2008. doi: 10.1109/ICDM.2008.22. URL https://doi.org/10.1109/ICDM.2008.22.

[28] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[29] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

[30] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263. URL https://doi.org/10.1109/MC.2009.263.

[31] W. Krichene and S. Rendle. On sampled metrics for item recommendation. In R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 1748–1757. ACM, 2020. doi: 10.1145/3394486.3403226. URL https://doi.org/10.1145/3394486.3403226.

[32] M. Kubat. Neural networks: a comprehensive foundation by simon haykin, macmillan, 1994, ISBN 0-02-352781-7. *Knowl. Eng. Rev.*, 13(4):409–412, 1999. doi: 10.1017/S0269888998214044. URL https://doi.org/10.1017/S0269888998214044.

[33] D. Lee, S. Kang, H. Ju, C. Park, and H. Yu. Bootstrapping user and item representations for one-class collaborative filtering. In F. Diaz, C. Shah, T. Suel, P. Castells, R. Jones, and T. Sakai, editors, *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval,*

*Virtual Event, Canada, July 11-15, 2021*, pages 1513–1522. ACM, 2021. doi: 10.1145/3404835.3462935. URL https://doi.org/10.1145/3404835.3462935.

[34] D. Lian, Q. Liu, and E. Chen. Personalized ranking with importance sampling. In Y. Huang, I. King, T. Liu, and M. van Steen, editors, *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 1093–1103. ACM / IW3C2, 2020. doi: 10.1145/3366423.3380187. URL https://doi.org/10.1145/3366423.3380187.

[35] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara. Variational autoencoders for collaborative filtering. In P. Champin, F. Gandon, M. Lalmas, and P. G. Ipeirotis, editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 689–698. ACM, 2018. doi: 10.1145/3178876.3186150. URL https://doi.org/10.1145/3178876.3186150.

[36] B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Syst. Appl.*, 41(4):2065–2073, 2014. doi: 10.1016/j.eswa.2013.09.005. URL https://doi.org/10.1016/j.eswa.2013.09.005.

[37] Z. Lin, C. Tian, Y. Hou, and W. X. Zhao. Improving graph collaborative filtering with neighborhood-enriched contrastive learning. In F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 2320–2329. ACM, 2022. doi: 10.1145/3485447.3512104. URL https://doi.org/10.1145/3485447.3512104.

[38] J. Liu, D. Li, H. Gu, T. Lu, P. Zhang, L. Shang, and N. Gu. Personalized graph signal processing for collaborative filtering. In Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, and G. Houben, editors, *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 1264–1272. ACM, 2023. doi: 10.1145/3543507.3583466. URL https://doi.org/10.1145/3543507.3583466.

[39] P. Lops, M. Degemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, 2011. URL https://api.semanticscholar.org/CorpusID:6102334.

[40] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[41] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In R. Baeza-Yates, M. Lalmas, A. Moffat, and B. A. Ribeiro-Neto, editors, *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 43–52. ACM, 2015. doi: 10.1145/2766462.2767755. URL https://doi.org/10.1145/2766462.2767755.

[42] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017. URL http://arxiv.org/abs/1703.00548.

[43] J. Ni, J. Li, and J. J. McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 188–197. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1018. URL https://doi.org/10.18653/v1/D19-1018.

[44] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. M. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 502–511. IEEE Computer Society, 2008. doi: 10.1109/ICDM.2008.16. URL https://doi.org/10.1109/ICDM.2008.16.

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[46] B. Paudel, F. Christoffel, C. Newell, and A. Bernstein. Updatable, accurate, diverse, and scalable recommendations for interactive applications. *ACM Trans. Interact. Intell. Syst.*, 7(1):1:1–1:34, 2017. doi: 10.1145/2955101. URL https://doi.org/10.1145/2955101.

[47] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The Adaptive Web*, 2007. URL https://api.semanticscholar.org/CorpusID:7364032.

[48] H. Petzka, A. Fischer, and D. Lukovnikov. On the regularization of wasserstein gans. *CoRR*, abs/1709.08894, 2017. URL http://arxiv.org/abs/1709.08894.

[49] V. Rakesh, S. Wang, K. Shu, and H. Liu. Linked variational autoencoders for inferring substitutable and supplementary items. In J. S. Culpepper, A. Moffat, P. N. Bennett, and K. Lerman, editors, *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, pages 438–446. ACM, 2019. doi: 10.1145/3289600.3290963. URL https://doi.org/10.1145/3289600.3290963.

[50] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1410. URL https://doi.org/10.18653/v1/D19-1410.

[51] S. Rendle and C. Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In B. Carterette, F. Diaz, C. Castillo, and D. Metzler, editors, *Seventh ACM International Conference on Web Search and Data Mining, WSDM 2014, New York, NY, USA, February 24-28, 2014*, pages 273–282. ACM, 2014. doi: 10.1145/2556195.2556248. URL https://doi.org/10.1145/2556195.2556248.

[52] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In J. A. Bilmes and A. Y. Ng, editors, *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 452–461. AUAI Press, 2009. URL https://www.auai.org/uai2009/papers/UAI2009_0139_48141db02b9f0b02bc7158819ebfa2c7.pdf.

[53] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. *CoRR*, abs/1205.2618, 2012. URL http://arxiv.org/abs/1205.2618.

[54] S. Rendle, L. Zhang, and Y. Koren. On the difficulty of evaluating baselines: A study

on recommender systems. *CoRR*, abs/1905.01395, 2019. URL http://arxiv.org/abs/1905.01395.

[55] F. Ricci, L. Rokach, and B. Shapira, editors. *Recommender Systems Handbook*. Springer US, 2022. ISBN 978-1-0716-2196-7. doi: 10.1007/978-1-0716-2197-4. URL https://doi.org/10.1007/978-1-0716-2197-4.

[56] M. Rossetti, F. Stella, and M. Zanker. Contrasting offline and online results when evaluating recommendation algorithms. In S. Sen, W. Geyer, J. Freyne, and P. Castells, editors, *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, pages 31–34. ACM, 2016. doi: 10.1145/2959100.2959176. URL https://doi.org/10.1145/2959100.2959176.

[57] N. Sachdeva, G. Manco, E. Ritacco, and V. Pudi. Sequential variational autoencoders for collaborative filtering. In J. S. Culpepper, A. Moffat, P. N. Bennett, and K. Lerman, editors, *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*, pages 600–608. ACM, 2019. doi: 10.1145/3289600.3291007. URL https://doi.org/10.1145/3289600.3291007.

[58] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In V. Y. Shen, N. Saito, M. R. Lyu, and M. E. Zurko, editors, *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 285–295. ACM, 2001. doi: 10.1145/371920.372071. URL https://doi.org/10.1145/371920.372071.

[59] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In A. Gangemi, S. Leonardi, and A. Panconesi, editors, *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, pages 111–112. ACM, 2015. doi: 10.1145/2740908.2742726. URL https://doi.org/10.1145/2740908.2742726.

[60] O. S. Shalom, S. Berkovsky, R. Ronen, E. Ziklik, and A. Amir. Data quality matters in recommender systems. In H. Werthner, M. Zanker, J. Golbeck, and G. Semeraro, editors, *Proceedings of the 9th ACM Conference on Recommender Systems, RecSys 2015, Vienna, Austria, September 16-20, 2015*, pages 257–260. ACM, 2015. doi: 10.1145/2792838.2799670. URL https://doi.org/10.1145/2792838.2799670.

[61] Y. Shen, Y. Wu, Y. Zhang, C. Shan, J. Zhang, K. B. Letaief, and D. Li. How powerful is graph convolution for recommendation? In G. Demartini, G. Zuccon, J. S.

Culpepper, Z. Huang, and H. Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 1619–1629. ACM, 2021. doi: 10.1145/3459637.3482264. URL https://doi.org/10.1145/3459637.3482264.

[62] I. Shenbin, A. Alekseev, E. Tutubalina, V. Malykh, and S. I. Nikolenko. Recvae: A new variational autoencoder for top-n recommendations with implicit feedback. In J. Caverlee, X. B. Hu, M. Lalmas, and W. Wang, editors, *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 528–536. ACM, 2020. doi: 10.1145/3336191.3371831. URL https://doi.org/10.1145/3336191.3371831.

[63] H. Steck, L. Baltrunas, E. Elahi, D. Liang, Y. Raimond, and J. Basilico. Deep learning for recommender systems: A netflix case study. *AI Mag.*, 42(3):7–18, 2021. doi: 10.1609/AIMAG.V42I3.18140. URL https://doi.org/10.1609/aimag.v42i3.18140.

[64] Z. Sun, Z. Deng, J. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ.

[65] Z. Tao, X. Liu, Y. Xia, X. Wang, L. Yang, X. Huang, and T.-S. Chua. Self-supervised learning for multimedia recommendation. *IEEE Transactions on Multimedia*, 2022. doi: 10.1109/TMM.2022.3187556.

[66] Y. Tian, C. Zhang, Z. Guo, C. Huang, R. A. Metoyer, and N. V. Chawla. Reciperec: A heterogeneous graph learning model for recipe recommendation. In L. D. Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 3466–3472. ijcai.org, 2022. doi: 10.24963/IJCAI.2022/481. URL https://doi.org/10.24963/ijcai.2022/481.

[67] R. van den Berg, T. N. Kipf, and M. Welling. Graph convolutional matrix completion. *CoRR*, abs/1706.02263, 2017. URL http://arxiv.org/abs/1706.02263.

[68] R. van Meteren. Using content-based filtering for recommendation. 2000.

[69] S. Vargas and P. Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In B. Mobasher, R. D. Burke, D. Jannach, and G. Adomavicius, editors, *Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011*, pages 109–116. ACM, 2011. URL https://dl.acm.org/citation.cfm?id=2043955.

[70] Q. Wang, Y. Wei, J. Yin, J. Wu, X. Song, and L. Nie. Dualgnn: Dual graph neural network for multimedia recommendation. *IEEE Trans. Multim.*, 25:1074–1084, 2023. doi: 10.1109/TMM.2021.3138298. URL https://doi.org/10.1109/TMM.2021.3138298.

[71] X. Wang, X. He, M. Wang, F. Feng, and T. Chua. Neural graph collaborative filtering. In B. Piwowarski, M. Chevalier, É. Gaussier, Y. Maarek, J. Nie, and F. Scholer, editors, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 165–174. ACM, 2019. doi: 10.1145/3331184.3331267. URL https://doi.org/10.1145/3331184.3331267.

[72] X. Wang, H. Jin, A. Zhang, X. He, T. Xu, and T. Chua. Disentangled graph collaborative filtering. In J. X. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, and Y. Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 1001–1010. ACM, 2020. doi: 10.1145/3397271.3401137. URL https://doi.org/10.1145/3397271.3401137.

[73] W. Wei, C. Huang, L. Xia, and C. Zhang. Multi-modal self-supervised learning for recommendation. In Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, and G. Houben, editors, *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 790–800. ACM, 2023. doi: 10.1145/3543507.3583206. URL https://doi.org/10.1145/3543507.3583206.

[74] Y. Wei, X. Wang, L. Nie, X. He, R. Hong, and T. Chua. MMGCN: multi-modal graph convolution network for personalized recommendation of micro-video. In L. Amsaleg, B. Huet, M. A. Larson, G. Gravier, H. Hung, C. Ngo, and W. T. Ooi, editors, *Proceedings of the 27th ACM International Conference on Multimedia, MM 2019, Nice, France, October 21-25, 2019*, pages 1437–1445. ACM, 2019. doi: 10.1145/3343031.3351034. URL https://doi.org/10.1145/3343031.3351034.

[75] Y. Wei, X. Wang, L. Nie, X. He, and T. Chua. Graph-refined convolutional network for multimedia recommendation with implicit feedback. In C. W. Chen, R. Cucchiara, X. Hua, G. Qi, E. Ricci, Z. Zhang, and R. Zimmermann, editors, *MM '20: The 28th ACM International Conference on Multimedia, Virtual Event / Seattle, WA, USA, October 12-16, 2020*, pages 3541–3549. ACM, 2020. doi: 10.1145/3394171.3413556. URL https://doi.org/10.1145/3394171.3413556.

[76] Y. Wei, X. Wang, Q. Li, L. Nie, Y. Li, X. Li, and T. Chua. Contrastive learning for

cold-start recommendation. In H. T. Shen, Y. Zhuang, J. R. Smith, Y. Yang, P. César, F. Metze, and B. Prabhakaran, editors, *MM '21: ACM Multimedia Conference, Virtual Event, China, October 20 - 24, 2021*, pages 5382–5390. ACM, 2021. doi: 10.1145/3474085.3475665. URL https://doi.org/10.1145/3474085.3475665.

[77] Y. Wei, X. Wang, X. He, L. Nie, Y. Rui, and T. Chua. Hierarchical user intent graph network for multimedia recommendation. *IEEE Trans. Multim.*, 24:2701–2712, 2022. doi: 10.1109/TMM.2021.3088307. URL https://doi.org/10.1109/TMM.2021.3088307.

[78] J. Weston, S. Bengio, and N. Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Mach. Learn.*, 81(1):21–35, 2010. doi: 10.1007/S10994-010-5198-3. URL https://doi.org/10.1007/s10994-010-5198-3.

[79] J. Wu, X. Wang, F. Feng, X. He, L. Chen, J. Lian, and X. Xie. Self-supervised graph learning for recommendation. In F. Diaz, C. Shah, T. Suel, P. Castells, R. Jones, and T. Sakai, editors, *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 726–735. ACM, 2021. doi: 10.1145/3404835.3462862. URL https://doi.org/10.1145/3404835.3462862.

[80] L. Xia, C. Huang, Y. Xu, J. Zhao, D. Yin, and J. X. Huang. Hypergraph contrastive collaborative filtering. In E. Amigó, P. Castells, J. Gonzalo, B. Carterette, J. S. Culpepper, and G. Kazai, editors, *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 70–79. ACM, 2022. doi: 10.1145/3477495.3532058. URL https://doi.org/10.1145/3477495.3532058.

[81] L. Xia, C. Huang, C. Huang, K. Lin, T. Yu, and B. Kao. Automated self-supervised learning for recommendation. In Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, and G. Houben, editors, *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 992–1002. ACM, 2023. doi: 10.1145/3543507.3583336. URL https://doi.org/10.1145/3543507.3583336.

[82] C. Yang, Q. Wu, J. Jin, X. Gao, J. Pan, and G. Chen. Trading hard negatives and true negatives: A debiased contrastive collaborative filtering approach. In L. D. Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*,

pages 2355–2361. ijcai.org, 2022. doi: 10.24963/IJCAI.2022/327. URL https://doi.org/10.24963/ijcai.2022/327.

[83] W. Yang, K. Lu, P. Yang, and J. Lin. Critically examining the "neural hype": Weak baselines and the additivity of effectiveness gains from neural ranking models. In B. Piwowarski, M. Chevalier, É. Gaussier, Y. Maarek, J. Nie, and F. Scholer, editors, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 1129–1132. ACM, 2019. doi: 10.1145/3331184.3331340. URL https://doi.org/10.1145/3331184.3331340.

[84] T. Yao, X. Yi, D. Z. Cheng, F. X. Yu, T. Chen, A. K. Menon, L. Hong, E. H. Chi, S. Tjoa, J. J. Kang, and E. Ettinger. Self-supervised learning for large-scale item recommendations. In G. Demartini, G. Zuccon, J. S. Culpepper, Z. Huang, and H. Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 4321–4330. ACM, 2021. doi: 10.1145/3459637.3481952. URL https://doi.org/10.1145/3459637.3481952.

[85] Z. Yi, X. Wang, I. Ounis, and C. MacDonald. Multi-modal graph contrastive learning for micro-video recommendation. In E. Amigó, P. Castells, J. Gonzalo, B. Carterette, J. S. Culpepper, and G. Kazai, editors, *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 1807–1811. ACM, 2022. doi: 10.1145/3477495.3532027. URL https://doi.org/10.1145/3477495.3532027.

[86] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In Y. Guo and F. Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 974–983. ACM, 2018. doi: 10.1145/3219819.3219890. URL https://doi.org/10.1145/3219819.3219890.

[87] J. Zhang, X. Shi, S. Zhao, and I. King. STAR-GCN: stacked and reconstructed graph convolutional networks for recommender systems. In S. Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 4264–4270. ijcai.org, 2019. doi: 10.24963/ijcai.2019/592. URL https://doi.org/10.24963/ijcai.2019/592.

[88] J. Zhang, Y. Zhu, Q. Liu, S. Wu, S. Wang, and L. Wang. Mining latent structures

for multimedia recommendation. In H. T. Shen, Y. Zhuang, J. R. Smith, Y. Yang, P. César, F. Metze, and B. Prabhakaran, editors, *MM '21: ACM Multimedia Conference, Virtual Event, China, October 20 - 24, 2021*, pages 3872–3880. ACM, 2021. doi: 10.1145/3474085.3475259. URL https://doi.org/10.1145/3474085.3475259.

[89] W. Zhang, T. Chen, J. Wang, and Y. Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In G. J. F. Jones, P. Sheridan, D. Kelly, M. de Rijke, and T. Sakai, editors, *The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '13, Dublin, Ireland - July 28 - August 01, 2013*, pages 785–788. ACM, 2013. doi: 10.1145/2484028.2484126. URL https://doi.org/10.1145/2484028.2484126.

[90] T. Zhou, Z. Kuscsik, J.-G. Liu, M. Medo, J. R. Wakeling, and Y.-C. Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, Feb. 2010. ISSN 1091-6490. doi: 10.1073/pnas.1000488107. URL http://dx.doi.org/10.1073/pnas.1000488107.

[91] X. Zhou, H. Zhou, Y. Liu, Z. Zeng, C. Miao, P. Wang, Y. You, and F. Jiang. Bootstrap latent representations for multi-modal recommendation. In Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, and G. Houben, editors, *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 845–854. ACM, 2023. doi: 10.1145/3543507.3583251. URL https://doi.org/10.1145/3543507.3583251.

# A | Full Experimental Results

To avoid crowding Chapter 4, in this appendix, we report the full experiment results for each analyzed algorithm and for each dataset used. We included at the end also the hyperparameter ranges for the baselines.

We reported the following tables for each dataset of each reproduced algorithm:

- Results on different cutoffs.
- Results on all metrics at the cutoff used to optimize the model.
- Results on the beyond accuracy metrics at the cutoff used to optimize the model.
- Computation time report.

### A.0.1. Automated Self-Supervised Learning for Recommendation (AutoCF)

Table A.1: Experimental results for the AutoCF method on the Amazon dataset on different cutoffs.

| | @ 5 | | @ 10 | | @ 50 | | @ 100 | |
|---|---|---|---|---|---|---|---|---|
| | REC | NDCG | REC | NDCG | REC | NDCG | REC | NDCG |
| TopPop | 0.0047 | 0.0055 | 0.0074 | 0.0063 | 0.0237 | 0.0114 | 0.0387 | 0.0153 |
| ItemKNN CF cosine | **0.0998** | **0.1197** | **0.1345** | **0.1264** | **0.2282** | **0.1548** | **0.2719** | **0.1666** |
| RP3beta | **0.0988** | **0.1203** | **0.1355** | **0.1276** | **0.2364** | **0.1581** | **0.2791** | **0.1696** |
| GraphFilter CF | **0.0689** | **0.0821** | **0.1001** | **0.0898** | **0.1977** | **0.1198** | 0.2509 | **0.1339** |
| MF BPR | 0.0430 | 0.0501 | 0.0608 | 0.0549 | 0.1240 | 0.0748 | 0.1640 | 0.0855 |
| PureSVD | 0.0398 | 0.0535 | 0.0564 | 0.0558 | 0.1158 | 0.0733 | 0.1550 | 0.0837 |
| AutoCF paper | - | - | - | - | - | - | - | - |
| AutoCF | 0.0649 | 0.0762 | 0.0940 | 0.0839 | 0.1961 | 0.1156 | 0.2555 | 0.1311 |
| AutoCF no earlystopping | 0.0636 | 0.0742 | 0.0937 | 0.0824 | 0.1957 | 0.1142 | 0.2563 | 0.1301 |

Table A.2: Experimental results for the AutoCF method on the Gowalla dataset on different cutoffs.

| | @ 5 | | @ 10 | | @ 50 | | @ 100 | |
|---|---|---|---|---|---|---|---|---|
| | REC | NDCG | REC | NDCG | REC | NDCG | REC | NDCG |
| TopPop | 0.0117 | 0.0160 | 0.0211 | 0.0181 | 0.0526 | 0.0272 | 0.0748 | 0.0329 |
| ItemKNN CF cosine | **0.1289** | **0.1405** | **0.1857** | **0.1539** | 0.3904 | **0.2122** | 0.4933 | **0.2374** |
| RP3beta | **0.1360** | **0.1476** | **0.2009** | **0.1644** | **0.4095** | **0.2243** | **0.5153** | **0.2505** |
| GraphFilter CF | 0.1238 | 0.1302 | **0.1848** | 0.1466 | **0.4036** | 0.2096 | **0.5164** | **0.2374** |
| MF BPR | 0.0979 | 0.1039 | 0.1442 | 0.1160 | 0.3145 | 0.1649 | 0.4096 | 0.1881 |
| PureSVD | 0.0703 | 0.0835 | 0.1061 | 0.0914 | 0.2492 | 0.1321 | 0.3422 | 0.1545 |
| AutoCF paper | - | - | - | - | - | - | - | - |
| AutoCF | 0.1252 | 0.1345 | 0.1841 | 0.1493 | 0.3961 | 0.2100 | 0.5053 | 0.2365 |
| AutoCF no earlystopping | 0.1257 | 0.1366 | 0.1831 | 0.1509 | 0.3905 | 0.2102 | 0.4993 | 0.2367 |

Table A.3: Experimental results for the AutoCF method on the Yelp dataset on different cutoffs.

| | @ 5 | | @ 10 | | @ 50 | | @ 100 | |
|---|---|---|---|---|---|---|---|---|
| | REC | NDCG | REC | NDCG | REC | NDCG | REC | NDCG |
| TopPop | 0.0081 | 0.0061 | 0.0139 | 0.0082 | 0.0395 | 0.0147 | 0.0620 | 0.0190 |
| ItemKNN CF cosine | **0.0332** | **0.0267** | **0.0510** | **0.0329** | 0.1199 | **0.0510** | 0.1647 | 0.0599 |
| RP3beta | **0.0313** | **0.0251** | **0.0472** | **0.0306** | 0.1147 | 0.0483 | 0.1555 | 0.0565 |
| GraphFilter CF | **0.0295** | **0.0227** | **0.0485** | **0.0294** | **0.1328** | **0.0510** | **0.1954** | **0.0632** |
| MF BPR | 0.0147 | 0.0118 | 0.0235 | 0.0148 | 0.0640 | 0.0254 | 0.0945 | 0.0314 |
| PureSVD | 0.0214 | 0.0174 | 0.0333 | 0.0215 | 0.0915 | 0.0366 | 0.1327 | 0.0447 |
| AutoCF paper | - | - | - | - | - | - | - | - |
| AutoCF | 0.0275 | 0.0219 | 0.0448 | 0.0281 | 0.1266 | 0.0491 | 0.1903 | 0.0615 |
| AutoCF no earlystopping | 0.0231 | 0.0178 | 0.0362 | 0.0224 | 0.0962 | 0.0378 | 0.1365 | 0.0459 |

Table A.4: Experimental results for the AutoCF method on the Amazon dataset on all available metrics.

| | @ 20 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0033 | 0.0122 | 0.0008 | 0.0134 | 0.0079 | 0.0052 | 0.0517 |
| ItemKNN CF cosine | **0.0405** | **0.1735** | **0.0193** | **0.2109** | **0.1381** | **0.0657** | **0.4302** |
| RP3beta | **0.0424** | **0.1777** | **0.0201** | **0.2130** | **0.1403** | **0.0685** | **0.4373** |
| GraphFilter CF | **0.0329** | **0.1387** | **0.0139** | **0.1496** | **0.1019** | **0.0532** | 0.3579 |
| MF BPR | 0.0194 | 0.0835 | 0.0075 | 0.0967 | 0.0624 | 0.0314 | 0.2482 |
| PureSVD | 0.0209 | 0.0781 | 0.0089 | 0.1062 | 0.0619 | 0.0329 | 0.2502 |
| AutoCF paper | - | 0.1277 | - | - | 0.0879 | - | - |
| AutoCF | 0.0307 | 0.1312 | 0.0122 | 0.1437 | 0.0959 | 0.0498 | 0.3586 |
| AutoCF no earlystopping | 0.0305 | 0.1315 | 0.0119 | 0.1408 | 0.0947 | 0.0495 | 0.3591 |

Table A.5: Experimental results for the AutoCF method on the Gowalla dataset on all available metrics.

| | @ 20 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0083 | 0.0327 | 0.0024 | 0.0380 | 0.0215 | 0.0132 | 0.1280 |
| ItemKNN CF cosine | **0.0498** | **0.2640** | **0.0211** | **0.2380** | **0.1771** | **0.0838** | **0.5499** |
| RP3beta | **0.0525** | **0.2795** | **0.0220** | **0.2509** | **0.1878** | **0.0884** | **0.5745** |
| GraphFilter CF | **0.0491** | **0.2677** | 0.0192 | 0.2242 | 0.1716 | **0.0830** | **0.5544** |
| MF BPR | 0.0377 | 0.2074 | 0.0143 | 0.1856 | 0.1350 | 0.0638 | 0.4649 |
| PureSVD | 0.0340 | 0.1569 | 0.0134 | 0.1555 | 0.1062 | 0.0559 | 0.4007 |
| AutoCF paper | - | 0.2538 | - | - | 0.1645 | - | - |
| AutoCF | 0.0488 | 0.2629 | 0.0197 | 0.2314 | 0.1729 | 0.0823 | 0.5491 |
| AutoCF no earlystopping | 0.0485 | 0.2610 | 0.0199 | 0.2364 | 0.1742 | 0.0818 | 0.5472 |

Table A.6: Experimental results for the AutoCF method on the Yelp dataset on all available metrics.

| | @ 20 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0021 | 0.0229 | 0.0005 | 0.0097 | 0.0108 | 0.0038 | 0.0391 |
| ItemKNN CF cosine | **0.0079** | **0.0753** | **0.0024** | **0.0418** | **0.0402** | **0.0143** | **0.1353** |
| RP3beta | **0.0076** | 0.0715 | **0.0023** | **0.0396** | **0.0379** | **0.0137** | 0.1302 |
| GraphFilter CF | **0.0076** | **0.0759** | **0.0020** | **0.0365** | **0.0375** | **0.0139** | **0.1346** |
| MF BPR | 0.0041 | 0.0371 | 0.0011 | 0.0199 | 0.0189 | 0.0073 | 0.0730 |
| PureSVD | 0.0055 | 0.0521 | 0.0016 | 0.0287 | 0.0272 | 0.0100 | 0.0984 |
| AutoCF paper | - | **0.0869** | - | - | **0.0437** | - | - |
| AutoCF | 0.0074 | 0.0731 | 0.0020 | 0.0362 | 0.0364 | 0.0135 | 0.1310 |
| AutoCF no earlystopping | 0.0058 | 0.0563 | 0.0016 | 0.0284 | 0.0283 | 0.0105 | 0.1040 |

Table A.7: Experimental results for the AutoCF method on the Gowalla dataset on beyond accuracy metrics.

| | @ 20 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0095 | 0.0400 | 0.0018 | 0.0015 | 0.0011 | 4.4239 |
| ItemKNN CF cosine | 0.0133 | 0.9881 | **0.8696** | 0.3147 | 0.2409 | 12.3011 |
| RP3beta | 0.0132 | 0.9860 | **0.8768** | **0.3285** | 0.2583 | 12.2988 |
| GraphFilter CF | **0.0136** | 0.9904 | **0.8788** | 0.3148 | **0.3004** | **12.6412** |
| MF BPR | 0.0128 | 0.9861 | **0.7165** | 0.2200 | 0.1661 | 11.8271 |
| PureSVD | 0.0129 | 0.9885 | 0.3365 | 0.1751 | 0.0956 | 11.3072 |
| AutoCF paper | - | - | - | - | - | - |
| AutoCF | 0.0133 | 0.9903 | 0.6658 | 0.3124 | 0.2432 | 12.4389 |
| AutoCF no earlystopping | 0.0134 | **0.9909** | 0.7114 | 0.3224 | 0.2639 | 12.5650 |

Table A.8: Experimental results for the AutoCF method on the Yelp dataset on beyond accuracy metrics.

| | @ 20 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0075 | 0.0082 | 0.0009 | 0.0009 | 0.0008 | 4.3480 |
| ItemKNN CF cosine | 0.0096 | 0.9843 | **0.9093** | 0.0656 | 0.1960 | 12.1753 |
| RP3beta | 0.0096 | 0.9834 | **0.8678** | 0.0605 | 0.1801 | 12.0775 |
| GraphFilter CF | 0.0093 | 0.9518 | 0.6633 | 0.0358 | 0.0871 | 10.5471 |
| MF BPR | 0.0091 | 0.9824 | 0.4552 | 0.0380 | 0.0743 | 11.2135 |
| PureSVD | 0.0088 | 0.9499 | 0.0632 | 0.0254 | 0.0161 | 9.1833 |
| AutoCF paper | - | - | - | - | - | - |
| AutoCF | 0.0089 | 0.9585 | 0.1320 | 0.0294 | 0.0208 | 9.5348 |
| AutoCF no earlystopping | **0.0102** | **0.9964** | 0.7565 | **0.0722** | **0.2922** | **13.2885** |

Table A.9: Computation time for the algorithms in the selected results for the AutoCF method on the Gowalla dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.01 [sec] | 62.02 [sec] / 1.03 [min] | 359 |
| ItemKNN CF cosine | 2.60 ± 0.24 [sec] | 68.14 [sec] / 1.14 ± 0.02 [min] | 321 |
| RP3beta | 14.24 ± 4.56 [sec] | 69.89 [sec] / 1.16 ± 0.07 [min] | 318 |
| GraphFilter CF | 66.56 [sec] / 1.11 ± 0.46 [min] | 70.57 [sec] / 1.18 ± 0.18 [min] | 407 |
| MF BPR | 1121.06 [sec] / 18.68 ± 13.35 [min] | 53.07 ± 9.29 [sec] | 468 |
| PureSVD | 4.16 ± 2.04 [sec] | 73.05 [sec] / 1.22 ± 0.09 [min] | 268 |
| AutoCF paper | - | - | - |
| AutoCF | 8319.65 [sec] / 2.31 [hour] | 52.70 [sec] | 422 |
| AutoCF no earlystopping | 13757.91 [sec] / 3.82 [hour] | 46.86 [sec] | 475 |

Table A.10: Computation time for the algorithms in the selected results for the AutoCF method on the Yelp dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.00 [sec] | 87.87 [sec] / 1.46 [min] | 349 |
| ItemKNN CF cosine | 3.95 ± 0.31 [sec] | 87.53 [sec] / 1.46 ± 0.03 [min] | 343 |
| RP3beta | 7.57 ± 0.98 [sec] | 86.32 [sec] / 1.44 ± 0.03 [min] | 358 |
| GraphFilter CF | 93.45 [sec] / 1.56 ± 0.58 [min] | 80.73 [sec] / 1.35 ± 0.16 [min] | 415 |
| MF BPR | 564.09 [sec] / 9.40 ± 10.38 [min] | 76.29 [sec] / 1.27 ± 0.18 [min] | 436 |
| PureSVD | 3.39 ± 1.93 [sec] | 100.73 [sec] / 1.68 ± 0.11 [min] | 289 |
| AutoCF paper | - | - | - |
| AutoCF | 2297.66 [sec] / 38.29 [min] | 63.70 [sec] / 1.06 [min] | 481 |
| AutoCF no earlystopping | 7217.61 [sec] / 2.00 [hour] | 64.43 [sec] / 1.07 [min] | 475 |

### A.0.2. Bootstrap Latent Representations for Multi-modal Recommendation (BM3)

Table A.11: Experimental results for the BM3 method on the Amazon Baby dataset on different cutoffs.

| | @ 5 | | @ 10 | | @ 50 | | @ 100 | |
|---|---|---|---|---|---|---|---|---|
| | REC | NDCG | REC | NDCG | REC | NDCG | REC | NDCG |
| TopPop | 0.0199 | 0.0130 | 0.0299 | 0.0162 | 0.0951 | 0.0302 | 0.1507 | 0.0393 |
| ItemKNN CF cosine | **0.0415** | **0.0287** | **0.0631** | **0.0357** | 0.1519 | **0.0551** | 0.2159 | **0.0656** |
| RP3beta | **0.0460** | **0.0318** | **0.0673** | **0.0387** | 0.1560 | **0.0581** | 0.2156 | **0.0679** |
| GraphFilter CF | **0.0365** | **0.0251** | 0.0560 | **0.0314** | 0.1410 | 0.0500 | 0.2031 | 0.0602 |
| MF BPR | 0.0211 | 0.0144 | 0.0325 | 0.0180 | 0.0976 | 0.0321 | 0.1534 | 0.0412 |
| PureSVD | 0.0228 | 0.0157 | 0.0374 | 0.0204 | 0.1126 | 0.0367 | 0.1717 | 0.0463 |
| BM3 paper | - | - | 0.0564 | 0.0301 | - | - | - | - |
| BM3 | 0.0345 | 0.0233 | 0.0570 | 0.0307 | **0.1575** | 0.0527 | **0.2266** | 0.0642 |
| BM3 no earlystopping | 0.0261 | 0.0174 | 0.0415 | 0.0225 | 0.1078 | 0.0371 | 0.1578 | 0.0455 |

Table A.12: Experimental results for the BM3 method on the Amazon Sports dataset on different cutoffs.

| | @ 5 | | @ 10 | | @ 50 | | @ 100 | |
|---|---|---|---|---|---|---|---|---|
| | REC | NDCG | REC | NDCG | REC | NDCG | REC | NDCG |
| TopPop | 0.0113 | 0.0079 | 0.0172 | 0.0098 | 0.0506 | 0.0170 | 0.0795 | 0.0218 |
| ItemKNN CF cosine | **0.0552** | **0.0380** | **0.0781** | **0.0454** | 0.1596 | **0.0634** | 0.2076 | **0.0713** |
| RP3beta | **0.0581** | **0.0404** | **0.0832** | **0.0486** | 0.1661 | **0.0669** | 0.2149 | **0.0750** |
| GraphFilter CF | 0.0457 | 0.0305 | 0.0685 | 0.0380 | 0.1586 | 0.0578 | 0.2148 | 0.0670 |
| MF BPR | 0.0115 | 0.0069 | 0.0194 | 0.0095 | 0.0584 | 0.0178 | 0.0886 | 0.0228 |
| PureSVD | 0.0288 | 0.0201 | 0.0408 | 0.0241 | 0.0948 | 0.0359 | 0.1367 | 0.0427 |
| BM3 paper | - | - | 0.0656 | 0.0355 | - | - | - | - |
| BM3 | 0.0425 | 0.0284 | 0.0672 | 0.0365 | **0.1681** | 0.0589 | **0.2328** | 0.0696 |
| BM3 no earlystopping | 0.0461 | 0.0309 | 0.0701 | 0.0388 | **0.1673** | 0.0604 | **0.2305** | 0.0709 |

Table A.13: Experimental results for the BM3 method on the Amazon Baby dataset on all available metrics.

| | @ 20 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0026 | 0.0500 | 0.0007 | 0.0138 | 0.0213 | 0.0050 | 0.0524 |
| ItemKNN CF cosine | 0.0049 | **0.0922** | **0.0015** | **0.0302** | **0.0431** | 0.0093 | 0.0966 |
| RP3beta | **0.0051** | **0.0965** | **0.0017** | **0.0329** | **0.0462** | **0.0097** | **0.1011** |
| GraphFilter CF | 0.0045 | 0.0854 | **0.0013** | **0.0268** | **0.0389** | 0.0086 | 0.0902 |
| MF BPR | 0.0028 | 0.0537 | 0.0008 | 0.0154 | 0.0233 | 0.0053 | 0.0559 |
| PureSVD | 0.0032 | 0.0599 | 0.0009 | 0.0176 | 0.0261 | 0.0061 | 0.0634 |
| BM3 paper | - | 0.0883 | - | - | 0.0383 | - | - |
| BM3 | 0.0049 | 0.0888 | 0.0013 | 0.0264 | 0.0388 | 0.0093 | 0.0971 |
| BM3 no earlystopping | 0.0036 | 0.0638 | 0.0010 | 0.0194 | 0.0283 | 0.0068 | 0.0704 |

Table A.14: Experimental results for the BM3 method on the Amazon Sports dataset on all available metrics.

| | @ 20 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0015 | 0.0284 | 0.0004 | 0.0086 | 0.0126 | 0.0029 | 0.0303 |
| ItemKNN CF cosine | 0.0057 | **0.1079** | **0.0019** | **0.0384** | **0.0530** | 0.0108 | 0.1125 |
| RP3beta | **0.0061** | **0.1144** | **0.0021** | **0.0412** | **0.0565** | **0.0115** | **0.1195** |
| GraphFilter CF | 0.0053 | 0.1006 | 0.0016 | 0.0318 | 0.0462 | 0.0101 | 0.1053 |
| MF BPR | 0.0016 | 0.0309 | 0.0004 | 0.0075 | 0.0124 | 0.0031 | 0.0327 |
| PureSVD | 0.0032 | 0.0596 | 0.0011 | 0.0209 | 0.0288 | 0.0060 | 0.0628 |
| BM3 paper | - | 0.0980 | - | - | 0.0438 | - | - |
| BM3 | 0.0057 | 0.1027 | 0.0016 | 0.0314 | 0.0456 | 0.0108 | 0.1122 |
| BM3 no earlystopping | 0.0058 | 0.1045 | 0.0017 | 0.0335 | 0.0477 | 0.0110 | 0.1142 |

Table A.15: Computation time for the algorithms in the selected results for the BM3 method on the Amazon Sports dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.01 [sec] | 408.76 [sec] / 6.81 [min] | 87 |
| ItemKNN CF cosine | 2.20 ± 0.19 [sec] | 114.61 [sec] / 1.91 ± 0.02 [min] | 309 |
| RP3beta | 6.89 ± 1.09 [sec] | 111.35 [sec] / 1.86 ± 0.07 [min] | 310 |
| GraphFilter CF | 78.17 [sec] / 1.30 ± 0.57 [min] | 124.23 [sec] / 2.07 ± 0.08 [min] | 293 |
| MF BPR | 972.47 [sec] / 16.21 ± 16.44 [min] | 95.42 [sec] / 1.59 ± 0.19 [min] | 410 |
| PureSVD | 3.45 ± 1.87 [sec] | 119.14 [sec] / 1.99 ± 0.09 [min] | 280 |
| BM3 paper | - | - | - |
| BM3 | 961.87 [sec] / 16.03 [min] | 66.64 [sec] / 1.11 [min] | 534 |
| BM3 no earlystopping | 1664.58 [sec] / 27.74 [min] | 68.35 [sec] / 1.14 [min] | 521 |

### A.0.3. Multi-Modal Self-Supervised Learning for Recommendation (MMSSL)

Table A.16: Experimental results for the MMSSL method on the Amazon Baby dataset on different cutoffs.

| | @ 5 | | | @ 10 | | | @ 50 | | |
|---|---|---|---|---|---|---|---|---|---|
| | REC | PREC | NDCG | REC | PREC | NDCG | REC | PREC | NDCG |
| TopPop | 0.0181 | 0.0038 | 0.0114 | 0.0284 | 0.0030 | 0.0149 | 0.0932 | 0.0020 | 0.0288 |
| ItemKNN CF cosine | **0.0412** | **0.0086** | **0.0279** | **0.0631** | 0.0066 | **0.0349** | 0.1505 | 0.0032 | 0.0541 |
| RP3beta | 0.0385 | 0.0081 | **0.0267** | 0.0598 | 0.0063 | 0.0336 | 0.1464 | 0.0031 | 0.0525 |
| GraphFilter CF | 0.0363 | 0.0076 | 0.0242 | 0.0595 | 0.0063 | 0.0317 | 0.1560 | 0.0033 | 0.0528 |
| MF BPR | 0.0124 | 0.0026 | 0.0084 | 0.0186 | 0.0020 | 0.0104 | 0.0517 | 0.0011 | 0.0176 |
| PureSVD | 0.0246 | 0.0052 | 0.0160 | 0.0411 | 0.0044 | 0.0213 | 0.1186 | 0.0025 | 0.0381 |
| MMSSL paper | - | - | - | - | - | - | - | - | - |
| MMSSL | 0.0392 | 0.0083 | 0.0260 | 0.0628 | **0.0066** | 0.0336 | **0.1649** | **0.0035** | **0.0559** |
| MMSSL no earlystopping | 0.0221 | 0.0046 | 0.0146 | 0.0337 | 0.0035 | 0.0184 | 0.0872 | 0.0018 | 0.0301 |

Table A.17: Experimental results for the MMSSL method on the AllRecipes dataset on different cutoffs.

| | @ 5 | | | @ 10 | | | @ 50 | | |
|---|---|---|---|---|---|---|---|---|---|
| | REC | PREC | NDCG | REC | PREC | NDCG | REC | PREC | NDCG |
| TopPop | **0.0143** | **0.0029** | **0.0102** | **0.0281** | **0.0028** | **0.0145** | **0.0767** | **0.0015** | **0.0254** |
| ItemKNN CF cosine | 0.0105 | 0.0021 | 0.0067 | 0.0189 | 0.0019 | 0.0094 | 0.0551 | 0.0011 | 0.0171 |
| RP3beta | 0.0113 | 0.0023 | 0.0068 | 0.0179 | 0.0018 | 0.0089 | 0.0537 | 0.0011 | 0.0166 |
| GraphFilter CF | 0.0105 | 0.0021 | 0.0065 | 0.0194 | 0.0019 | 0.0094 | 0.0638 | 0.0013 | 0.0188 |
| MF BPR | 0.0118 | 0.0024 | 0.0063 | 0.0143 | 0.0014 | 0.0071 | 0.0545 | 0.0011 | 0.0154 |
| PureSVD | **0.0146** | **0.0029** | **0.0084** | **0.0249** | **0.0025** | **0.0117** | **0.0830** | **0.0017** | **0.0247** |
| MMSSL paper | - | - | - | - | - | - | - | - | - |
| MMSSL | 0.0118 | 0.0024 | 0.0075 | 0.0220 | 0.0022 | 0.0108 | 0.0719 | 0.0014 | 0.0214 |
| MMSSL no earlystopping | 0.0060 | 0.0012 | 0.0035 | 0.0100 | 0.0010 | 0.0048 | 0.0348 | 0.0007 | 0.0100 |

Table A.18: Experimental results for the MMSSL method on the Tiktok dataset on different cutoffs.

| | @ 5 | | | @ 10 | | | @ 50 | | |
|---|---|---|---|---|---|---|---|---|---|
| | REC | PREC | NDCG | REC | PREC | NDCG | REC | PREC | NDCG |
| TopPop | 0.0440 | 0.0088 | 0.0322 | 0.0476 | 0.0048 | 0.0333 | 0.1545 | 0.0031 | 0.0563 |
| ItemKNN CF cosine | **0.0515** | **0.0103** | **0.0353** | 0.0788 | 0.0079 | **0.0440** | 0.1633 | 0.0033 | 0.0625 |
| RP3beta | **0.0587** | **0.0117** | **0.0388** | **0.0853** | **0.0085** | **0.0473** | 0.1835 | 0.0037 | **0.0687** |
| GraphFilter CF | 0.0481 | 0.0096 | 0.0313 | 0.0739 | 0.0074 | 0.0396 | 0.1842 | 0.0037 | 0.0636 |
| MF BPR | 0.0192 | 0.0038 | 0.0126 | 0.0313 | 0.0031 | 0.0165 | 0.0778 | 0.0016 | 0.0268 |
| PureSVD | 0.0320 | 0.0064 | 0.0213 | 0.0426 | 0.0043 | 0.0248 | 0.0790 | 0.0016 | 0.0326 |
| MMSSL paper | - | - | - | - | - | - | - | - | - |
| MMSSL | 0.0514 | 0.0103 | 0.0334 | 0.0832 | 0.0083 | 0.0437 | **0.1954** | **0.0039** | 0.0679 |
| MMSSL no earlystopping | 0.0351 | 0.0070 | 0.0229 | 0.0553 | 0.0055 | 0.0294 | 0.1392 | 0.0028 | 0.0475 |

Table A.19: Experimental results for the MMSSL method on the Amazon Baby dataset on all available metrics.

| | @ 20 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0025 | 0.0464 | 0.0006 | 0.0123 | 0.0194 | 0.0047 | 0.0490 |
| ItemKNN CF cosine | 0.0049 | 0.0925 | **0.0015** | **0.0292** | **0.0424** | 0.0093 | 0.0966 |
| RP3beta | 0.0047 | 0.0886 | **0.0014** | **0.0283** | 0.0409 | 0.0089 | 0.0927 |
| GraphFilter CF | 0.0049 | 0.0929 | 0.0013 | 0.0263 | 0.0401 | 0.0093 | 0.0969 |
| MF BPR | 0.0015 | 0.0292 | 0.0004 | 0.0090 | 0.0132 | 0.0029 | 0.0309 |
| PureSVD | 0.0035 | 0.0652 | 0.0009 | 0.0176 | 0.0274 | 0.0066 | 0.0688 |
| MMSSL paper | 0.0051 | 0.0962 | - | - | 0.0422 | - | - |
| MMSSL | **0.0051** | **0.0970** | 0.0014 | 0.0279 | 0.0423 | **0.0097** | **0.1011** |
| MMSSL no earlystopping | 0.0027 | 0.0516 | 0.0008 | 0.0153 | 0.0230 | 0.0052 | 0.0543 |

Table A.20: Experimental results for the MMSSL method on the AllRecipes dataset on all available metrics.

| | @ 20 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | **0.0026** | **0.0511** | **0.0006** | **0.0122** | **0.0205** | **0.0049** | **0.0511** |
| ItemKNN CF cosine | 0.0015 | 0.0306 | 0.0004 | 0.0073 | 0.0123 | 0.0029 | 0.0306 |
| RP3beta | 0.0016 | 0.0313 | 0.0004 | 0.0070 | 0.0122 | 0.0030 | 0.0313 |
| GraphFilter CF | 0.0016 | 0.0313 | 0.0004 | 0.0072 | 0.0124 | 0.0030 | 0.0313 |
| MF BPR | 0.0012 | 0.0249 | 0.0003 | 0.0054 | 0.0095 | 0.0024 | 0.0249 |
| PureSVD | **0.0026** | **0.0523** | **0.0005** | **0.0097** | **0.0188** | **0.0050** | **0.0523** |
| MMSSL paper | 0.0018 | 0.0367 | - | - | 0.0135 | - | - |
| MMSSL | 0.0019 | 0.0382 | 0.0004 | 0.0085 | 0.0148 | 0.0036 | 0.0382 |
| MMSSL no earlystopping | 0.0009 | 0.0175 | 0.0002 | 0.0037 | 0.0066 | 0.0017 | 0.0175 |

Table A.21: Experimental results for the MMSSL method on the Tiktok dataset on all available metrics.

| | @ 20 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0040 | 0.0798 | 0.0016 | 0.0311 | 0.0416 | 0.0076 | 0.0798 |
| ItemKNN CF cosine | 0.0056 | 0.1117 | **0.0018** | **0.0358** | 0.0523 | 0.0106 | 0.1117 |
| RP3beta | 0.0060 | 0.1207 | **0.0019** | **0.0382** | **0.0563** | 0.0115 | 0.1207 |
| GraphFilter CF | 0.0057 | 0.1132 | 0.0016 | 0.0319 | 0.0495 | 0.0108 | 0.1132 |
| MF BPR | 0.0025 | 0.0502 | 0.0007 | 0.0134 | 0.0213 | 0.0048 | 0.0502 |
| PureSVD | 0.0027 | 0.0550 | 0.0010 | 0.0201 | 0.0279 | 0.0052 | 0.0550 |
| MMSSL paper | 0.0046 | 0.0921 | - | - | 0.0392 | - | - |
| MMSSL | **0.0061** | **0.1215** | 0.0017 | 0.0344 | 0.0533 | **0.0116** | **0.1215** |
| MMSSL no earlystopping | 0.0042 | 0.0850 | 0.0012 | 0.0235 | 0.0368 | 0.0081 | 0.0850 |

Table A.22: Experimental results for the MMSSL method on the AllRecipes dataset on beyond accuracy metrics.

| | @ 20 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0163 | 0.0190 | 0.0024 | 0.0021 | 0.0021 | 4.3672 |
| ItemKNN CF cosine | 0.0211 | 0.9182 | 0.7541 | **0.0088** | 0.1312 | 9.8993 |
| RP3beta | 0.0209 | 0.9024 | 0.7477 | **0.0086** | 0.1268 | 9.7619 |
| GraphFilter CF | 0.0190 | 0.7346 | 0.3133 | 0.0045 | 0.0298 | 7.6366 |
| MF BPR | 0.0175 | 0.5099 | 0.0046 | 0.0031 | 0.0040 | 5.3703 |
| PureSVD | 0.0165 | 0.0308 | 0.0046 | 0.0019 | 0.0021 | 4.4214 |
| MMSSL paper | - | - | - | - | - | - |
| MMSSL | 0.0180 | 0.6585 | 0.2901 | 0.0053 | 0.0197 | 7.1404 |
| MMSSL no earlystopping | **0.0227** | **0.9787** | **0.8395** | 0.0082 | **0.1980** | **11.1083** |

Table A.23: Experimental results for the MMSSL method on the Tiktok dataset on beyond accuracy metrics.

| | @ 20 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0253 | 0.0335 | 0.0057 | 0.0027 | 0.0032 | 4.4033 |
| ItemKNN CF cosine | 0.0307 | 0.9095 | 0.6955 | 0.0258 | 0.1018 | 9.2141 |
| RP3beta | 0.0306 | 0.9055 | 0.6338 | 0.0273 | 0.0994 | 9.2567 |
| GraphFilter CF | 0.0299 | 0.8102 | 0.3914 | 0.0176 | 0.0397 | 7.9156 |
| MF BPR | 0.0293 | 0.8764 | 0.3382 | 0.0110 | 0.0386 | 8.2226 |
| PureSVD | 0.0308 | 0.9493 | 0.3495 | 0.0110 | 0.0952 | 9.6347 |
| MMSSL paper | - | - | - | - | - | - |
| MMSSL | 0.0298 | 0.8718 | 0.4361 | 0.0225 | 0.0576 | 8.6081 |
| MMSSL no earlystopping | **0.0338** | **0.9736** | **0.8380** | **0.0300** | **0.2103** | **10.6629** |

Table A.24: Computation time for the algorithms in the selected results for the MMSSL method on the AllRecipes dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.00 [sec] | 16.66 [sec] | 469 |
| ItemKNN CF cosine | 0.47 ± 0.06 [sec] | 17.36 ± 0.83 [sec] | 469 |
| RP3beta | 1.12 ± 0.12 [sec] | 17.79 ± 1.08 [sec] | 408 |
| GraphFilter CF | 28.79 ± 12.70 [sec] | 18.62 ± 1.26 [sec] | 451 |
| MF BPR | 84.22 [sec] / 1.40 ± 2.14 [min] | 17.69 ± 1.17 [sec] | 409 |
| PureSVD | 1.14 ± 0.99 [sec] | 19.78 ± 2.10 [sec] | 337 |
| MMSSL paper | - | - | - |
| MMSSL | 679.56 [sec] / 11.33 [min] | 16.02 [sec] | 488 |
| MMSSL no earlystopping | 21451.33 [sec] / 5.96 [hour] | 15.59 [sec] | 502 |

Table A.25: Computation time for the algorithms in the selected results for the MMSSL method on the Tiktok dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.00 [sec] | 12.62 [sec] | 486 |
| ItemKNN CF cosine | 0.43 ± 0.06 [sec] | 14.07 ± 1.02 [sec] | 415 |
| RP3beta | 4.41 ± 2.41 [sec] | 14.40 ± 0.55 [sec] | 399 |
| GraphFilter CF | 18.12 ± 7.68 [sec] | 15.11 ± 1.36 [sec] | 458 |
| MF BPR | 53.75 ± 59.07 [sec] | 13.88 ± 2.15 [sec] | 360 |
| PureSVD | 0.90 ± 0.57 [sec] | 15.09 ± 1.71 [sec] | 325 |
| MMSSL paper | - | - | - |
| MMSSL | 740.13 [sec] / 12.34 [min] | 11.57 [sec] | 530 |
| MMSSL no earlystopping | 12326.41 [sec] / 3.42 [hour] | 11.45 [sec] | 535 |

### A.0.4. Fast Variational AutoEncoder with Inverted Multi-Index for Collaborative Filtering (FastVAE)

Table A.26: Experimental results for the FastVAE method on the MovieLens10M dataset on different cutoffs.

| | @ 5 | | @ 10 | | @ 20 | | @ 100 | |
|---|---|---|---|---|---|---|---|---|
| | NDCG | REC | NDCG | REC | NDCG | REC | NDCG | REC |
| TopPop | 0.1022 | 0.0679 | 0.1115 | 0.1107 | 0.1283 | 0.1665 | 0.1962 | 0.3809 |
| ItemKNN CF cosine | **0.2175** | **0.1428** | **0.2246** | **0.2142** | **0.2506** | 0.3074 | 0.3418 | 0.5961 |
| RP3beta | **0.2221** | **0.1468** | **0.2312** | **0.2226** | **0.2596** | **0.3221** | **0.3548** | 0.6215 |
| GraphFilter CF | **0.2382** | **0.1560** | **0.2471** | **0.2360** | **0.2760** | **0.3393** | **0.3684** | 0.6288 |
| MF BPR | 0.1873 | 0.1260 | 0.1995 | 0.1977 | 0.2290 | 0.2955 | 0.3237 | 0.5929 |
| PureSVD | 0.1888 | 0.1201 | 0.1940 | 0.1821 | 0.2168 | 0.2657 | 0.3020 | 0.5364 |
| FastVAE paper | - | - | - | - | - | - | - | - |
| FastVAE | 0.1921 | 0.1292 | 0.2064 | 0.2056 | 0.2377 | 0.3086 | 0.3386 | 0.6238 |
| FastVAE no earlystopping | 0.2011 | 0.1347 | 0.2143 | 0.2119 | 0.2456 | 0.3157 | 0.3462 | **0.6302** |

Table A.27: Experimental results for the FastVAE method on the Gowalla dataset on different cutoffs.

| | @ 5 | | @ 10 | | @ 20 | | @ 100 | |
|---|---|---|---|---|---|---|---|---|
| | NDCG | REC | NDCG | REC | NDCG | REC | NDCG | REC |
| TopPop | 0.0164 | 0.0119 | 0.0187 | 0.0207 | 0.0216 | 0.0303 | 0.0330 | 0.0692 |
| ItemKNN CF cosine | **0.1391** | **0.0975** | **0.1457** | **0.1441** | **0.1639** | **0.2066** | **0.2257** | **0.4208** |
| RP3beta | **0.1444** | **0.1026** | **0.1527** | **0.1531** | **0.1724** | **0.2194** | **0.2385** | **0.4462** |
| GraphFilter CF | **0.1383** | **0.0988** | **0.1466** | **0.1480** | **0.1668** | **0.2146** | **0.2343** | **0.4465** |
| MF BPR | 0.0928 | 0.0679 | 0.0993 | 0.1022 | 0.1149 | 0.1516 | 0.1663 | 0.3287 |
| PureSVD | **0.0998** | 0.0653 | **0.1029** | 0.0982 | 0.1153 | 0.1437 | 0.1653 | 0.3159 |
| FastVAE paper | - | - | - | - | - | - | - | - |
| FastVAE | 0.0903 | 0.0656 | 0.0973 | 0.1003 | 0.1134 | 0.1508 | 0.1696 | 0.3442 |
| FastVAE no earlystopping | 0.0939 | 0.0682 | 0.1011 | 0.1040 | 0.1180 | 0.1571 | 0.1758 | 0.3561 |

Table A.28: Experimental results for the FastVAE method on the MovieLens10M dataset on all available metrics.

| | @ 50 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0395 | 0.2716 | 0.0099 | 0.2226 | 0.1636 | 0.0689 | 0.7539 |
| ItemKNN CF cosine | 0.0643 | 0.4636 | 0.0218 | **0.4087** | 0.3018 | 0.1129 | 0.9163 |
| RP3beta | 0.0679 | 0.4866 | **0.0231** | **0.4134** | **0.3138** | 0.1191 | **0.9272** |
| GraphFilter CF | **0.0705** | **0.4992** | **0.0251** | **0.4348** | **0.3288** | **0.1235** | **0.9296** |
| MF BPR | 0.0636 | 0.4571 | 0.0197 | 0.3601 | 0.2826 | 0.1116 | 0.9112 |
| PureSVD | 0.0589 | 0.4113 | 0.0193 | 0.3633 | 0.2643 | 0.1030 | 0.8751 |
| FastVAE paper | - | **0.5078** | - | - | **0.3275** | - | - |
| FastVAE | 0.0677 | 0.4815 | 0.0210 | 0.3673 | 0.2953 | 0.1187 | 0.9207 |
| FastVAE no earlystopping | 0.0689 | 0.4882 | 0.0219 | 0.3794 | 0.3030 | 0.1208 | 0.9247 |

Table A.29: Experimental results for the FastVAE method on the Gowalla dataset on all available metrics.

| | @ 50 | | | | | | |
|---|---|---|---|---|---|---|---|
| | PREC | REC | MAP | MRR | NDCG | F1 | HR |
| TopPop | 0.0059 | 0.0493 | 0.0011 | 0.0443 | 0.0275 | 0.0106 | 0.2246 |
| ItemKNN CF cosine | **0.0371** | **0.3172** | **0.0119** | **0.2784** | **0.1974** | **0.0664** | **0.7564** |
| RP3beta | **0.0396** | **0.3382** | **0.0125** | **0.2890** | **0.2087** | **0.0709** | **0.7810** |
| GraphFilter CF | **0.0386** | **0.3355** | **0.0118** | **0.2807** | **0.2038** | **0.0692** | **0.7778** |
| MF BPR | 0.0272 | 0.2402 | 0.0071 | 0.2029 | 0.1422 | 0.0488 | 0.6594 |
| PureSVD | 0.0292 | 0.2289 | **0.0088** | **0.2091** | 0.1413 | 0.0518 | 0.6436 |
| FastVAE paper | - | **0.2971** | - | - | **0.1797** | - | - |
| FastVAE | 0.0287 | 0.2476 | 0.0074 | 0.1979 | 0.1433 | 0.0515 | 0.6700 |
| FastVAE no earlystopping | 0.0299 | 0.2566 | 0.0078 | 0.2034 | 0.1487 | 0.0535 | 0.6806 |

Table A.30: Experimental results for the FastVAE method on the Gowalla dataset on beyond accuracy metrics.

| | @ 50 | | | | | |
|---|---|---|---|---|---|---|
| | Novelty | Div. MIL | Cov. Item | Cov. Item Hit | Div. Gini | Div. Shannon |
| TopPop | 0.0127 | 0.0384 | 0.0020 | 0.0016 | 0.0013 | 5.7278 |
| ItemKNN CF cosine | **0.0168** | 0.9817 | 0.7616 | **0.3200** | 0.1605 | 12.8055 |
| RP3beta | 0.0167 | 0.9800 | **0.8009** | **0.3380** | 0.1718 | 12.8433 |
| GraphFilter CF | **0.0170** | 0.9826 | 0.7299 | **0.3306** | 0.1825 | 12.9811 |
| MF BPR | 0.0162 | 0.9726 | 0.6521 | 0.2004 | 0.1094 | 12.2014 |
| PureSVD | 0.0165 | 0.9836 | 0.3790 | 0.2060 | 0.0939 | 12.2944 |
| FastVAE paper | - | - | - | - | - | - |
| FastVAE | 0.0165 | 0.9807 | 0.6645 | 0.2529 | 0.1484 | 12.7058 |
| FastVAE no earlystopping | 0.0167 | **0.9851** | 0.7911 | 0.2915 | **0.2016** | **13.1342** |

Table A.31: Computation time for the algorithms in the selected results for the FastVAE method on the Gowalla dataset.

| | Train Time | Recommendation Time | Recommendation Throughput |
|---|---|---|---|
| TopPop | 0.01 [sec] | 98.99 [sec] / 1.65 [min] | 302 |
| ItemKNN CF cosine | 11.03 ± 0.63 [sec] | 108.15 [sec] / 1.80 ± 0.08 [min] | 269 |
| RP3beta | 44.28 ± 12.22 [sec] | 100.71 [sec] / 1.68 ± 0.09 [min] | 281 |
| GraphFilter CF | 161.06 [sec] / 2.68 ± 0.93 [min] | 113.16 [sec] / 1.89 ± 0.15 [min] | 277 |
| MF BPR | 4123.87 [sec] / 1.15 ± 0.82 [hour] | 82.02 [sec] / 1.37 ± 0.25 [min] | 445 |
| PureSVD | 6.72 ± 3.16 [sec] | 109.03 [sec] / 1.82 ± 0.08 [min] | 257 |
| FastVAE paper | - | - | - |
| FastVAE | 2285.06 [sec] / 38.08 [min] | 85.65 [sec] / 1.43 [min] | 349 |
| FastVAE no earlystopping | 3581.03 [sec] / 59.68 [min] | 85.96 [sec] / 1.43 [min] | 347 |

### A.0.5. Baselines hyperparameter values range

Table A.32: Hyperparameter values for our baselines.

| Algorithm | Hyperparameter | Range | Type | Distribution |
|---|---|---|---|---|
| ItemKNN cosine | topK | 5 - 1000 | Integer | uniform |
| | shrink | 0 - 1000 | Integer | uniform |
| | similarity | cosine | Categorical | |
| | normalize[a] | True, False | Categorical | |
| | feature weighting | none, TF-IDF, BM25 | Categorical | |
| RP3beta | topK | 5 - 1000 | Integer | uniform |
| | alpha | 0 - 2 | Real | uniform |
| | beta | 0 - 2 | Real | uniform |
| | normalize similarity[b] | True, False | Categorical | |
| GraphFilterCF | topK | 5 - min(5000, $n_{items}$-1) | Integer | uniform |
| | alpha | $10^{-3}$ - $10^{3}$ | Real | log-uniform |
| | num factors | 1 - 350 | Integer | uniform |
| MF BPR | num factors | 1 - 200[c] | Integer | uniform |
| | epochs | 1 - 1500 | Categorical | early-stopping |
| | sgd mode | sgd, adam, adagrad | Categorical | |
| | batch size | $2^{0}$ - $2^{10}$ | Categorical | log-uniform |
| | positive reg | $10^{-5}$ - $10^{-2}$ | Real | log-uniform |
| | negative reg | $10^{-5}$ - $10^{-2}$ | Real | log-uniform |
| | learning rate | $10^{-4}$ - $10^{-1}$ | Real | log-uniform |
| PureSVD | num factors | 1 - 350 | Integer | uniform |

[a]The *normalize* hyperparameter in KNNs refers to the use of the denominator when computing the similarity.

[b]The *normalize similarity* hyperparameter in P3alpha and RP3beta refers to applying L1 regularisation on the rows of the similarity matrix

[c]The number of factors is lower due to the algorithm being slower.

# List of Figures

# List of Tables