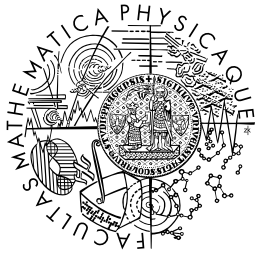


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Ladislav Maleček

Fairness in group recommender systems

Department of Software Engineering

Supervisor of the bachelor thesis:	Mgr. Ladislav Peška, Ph.D.
Study programme:	Informatics
Specialization:	Artificial Intelligence

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I dedicate this work to my closest friends and my family, whose never-ending support and kind words have kept me motivated throughout my studies and writing this thesis.

Furthermore, I want to thank my supervisor Mgr. Ladislav Peška, Ph.D., for his patience, time, and a considerable amount of valuable insight and suggestions, not only on a professional level.

Title: Fairness in group recommender systems

Author: Bc. Ladislav Maleček

Department: Department of Software Engineering

Supervisor: Mgr. Ladislav Peška, Ph.D., Department of Software Engineering

Abstract:

The goal of this thesis is to explore the area of group recommender systems with an emphasis on fairness. In the core part of our thesis, we have created a novel aggregation method that works on top of a single-user recommender system called Exactly Proportional Fuzzy D'Hondt. We have evaluated it on five datasets, in three different recommendation scenarios, and with two different types of artificially created groups. The proposed algorithm performed favorably with respect to several fairness metrics while maintaining a reasonable utility of the recommendation as well. Furthermore, we have created a set of tools to simplify the evaluation pipeline of group recommender systems. The main parts of the pipeline are a dataset downloader, matrix factorizer, and synthetic group creation scripts. We believe these tools may contribute towards more reproducible research in the group recommender systems domain.

Keywords: group recommender systems, fairness, synthetic datasets, preference aggregation

Contents

1. Introduction

The goal of this thesis is to explore the area of group recommender systems with an emphasis on fairness. Discuss fairness in the context of machine learning systems and, specifically, group recommender systems. Gather information about available datasets that can be used to evaluate group recommender systems. Design a method of artificial group creation for datasets that do not contain group information. Create a group recommendation algorithm inspired by the mandate allocation method D'Hondt. Finally, evaluate its performance compared to other related methods on a broad spectrum of different datasets.

We have created a novel aggregation method that works on top of a single-user recommender system called Exactly Proportional Fuzzy D'Hondt. We evaluated it on five datasets, three different recommendation scenarios, and two different types of artificially created groups. Further, we have created a set of tools that help future research with the evaluation and validation of new ideas in this domain on big datasets.

Most of us interact with many recommender systems daily, even if seemingly indirectly. The proliferation of this technology is astounding. Almost every interaction with today's web is in some way personalized. From search results, shopping, listening to music, reading news, browsing social media, and many more. Recommender systems have become quite literally — unavoidable.

We can view recommender systems from the following elementary perspective: they are algorithms that recommend items to users, where items and users can be many different things. For example, items can be movies, news articles, more complex objects, or even entire systems. Furthermore, users are real people or other entities that exhibit some preference on which the algorithm can decide.

One of the variants of recommender systems is when the recommendation results are shared among more users based on their shared (aggregated) preferences. This variant is a subset called group recommender systems. They are not as widely used as the non-group variants because we mostly use the web, listen to music and read the news as individuals, at least from the perspective of those systems. However, for some domains, there are valid use cases. We often listen to music and watch movies in groups. Select a restaurant and other public services not just for us. In these situations, group recommenders come in handy.

With groups as the target of a recommendation comes new challenges, one of them being how to measure satisfaction and ensure fairness among the group members. We first need to have a reliable way how to evaluate the recommendations. It becomes more complex than simply rating the results based on single feedback. Now, we have multiple users with possibly very different personal experiences, preferences, wants, and needs. We want to be fair towards all the individuals in the group. However, the fairness property can be tricky to describe and evaluate due to the subjective nature of preference perception and distribution among the group members.

Classical recommendation systems have been studied for quite a long time, but the group variant and more soft-level (meaning evaluation with metrics other than the classical accuracy and precision) thinking about them is pretty recent. With the rise of social dilemmas around recommender systems, the fairness-ensuring

topic is becoming more important than before. With that, there is a growing popularity of recommender systems that are trained (and therefore evaluated) with these novel requirements in mind.

1.1 Problem statement

The current research on the topic of group recommender systems is lacking. There are no standardized data sets that would offer an evaluation of the research without using various methods of data augmentation and artificial group creation. The augmentation strategies differ widely among research efforts. The evaluation process is often proprietary, slow, and designed only for small-scale datasets. Further, the definition of fairness is not unified. It can mean many different things and be evaluated with many different methods.

These problems mentioned above go hand in hand with the very subjective nature of user preference.

1.2 Research objective

We want to study how fairness can be defined in the context of recommender systems and how it can be measured and eventually used to improve recommendations in the group setting. Furthermore, we will explore different variants of fairness, such as long-term fairness and different distribution of fairness among group members.

The primary goal of this thesis is to research and design a novel group recommender system algorithm that would keep fairness as its primary optimization objective. We will try to adapt fairness-preserving methods, such as voting systems from other fields, to group recommendation problems. And then evaluate the new algorithm with already existing approaches in the domain of group recommender systems.

Additionally, we would like to research and contribute to data sets that could be used for the group setting. Expanding single-user data sets with data augmentation that would generate synthetic groups' information.

1.3 Thesis structure

We introduce single and group recommender systems in Chapter Recommender systems. Then we continue with a deep-dive into fairness in Chapter Fairness. Next, we introduce a selection of related algorithms used in the group recommender field in Chapter Related work. Following with an overview of single-user and multi-user datasets that are suitable for use in the group recommender domain and a method of creating synthetic group information from single-user dataset information in Chapter Datasets. Finally, we introduce our algorithm in Chapter Proposed group recommender system and define evaluation scenarios, describe experiments and discuss the results in Chapter Experiments.

2. Recommender systems

In this chapter, we briefly introduce what recommender systems are (hereinafter referred to as RS), then continue with a description of the group variant of recommender systems and introduce common approaches and methods they employ.

2.1 Recommender systems

Broadly speaking, recommender systems are algorithms that suggest items to users. From another perspective, they aim to predict how a user would rate (like) an unseen item. They are used in various settings, for instance, e-commerce, media consumption, social networks, expert systems, search engines, and many others.

At their core, as stated in [1] they are essentially an information filtering system that aims to select a subset of items by some filtration criteria. In this case, the criteria are the user's preference. RS become necessary when it comes to suppressing the explosive growth of information on the web and function as a defense system against overloading the user with the vast amount of data that is present in almost every system today.

They can be viewed as decision support systems that guide users in finding and identifying items based on their idea about the desired state. In this situation, the desired state is to find an item that they would like [2].

RS can provide both by filtering based on user preference and providing alternatives by utilizing similarity. In a way, finding a suitable item can be viewed as a collaboration between the user and the recommender system, with varying degrees of freedom, from passively accepting the RS recommendations to actively interacting by giving feedback and stating preferences.

2.1.1 High-level examples

Recommender systems are used in multiple ways. We now present high-level examples of where and how they are utilized the most.

- **Personalized merchandising**, where the system offers items that other users bought together with the viewed item. Items that a user could like based on the user's previous orders or viewed items.
- **Personalized content**, for content consumption services such as video and audio libraries. User is offered personalized content based on their preference profile, such as movies or videos that are similar to other content they consumed, globally popular for a regional subset of the user base, and so on.
- **Personalized news feed and social media feed**, offering users exciting content to keep them engaged with the service. In recent years there has been a push toward more socially responsible RS design in this context due to the overwhelming power of social media. It is important to deal with problems such as polarization [3], fairness and disagreement.

- **Expert systems**, helping doctors, operators, and other people to make informed decisions based on data. They can help to deal with data overload and filter relevant items and choices. As well as explore the item space when searching for solutions with only weakly defined requirements.
- **Search experience**, that considers previous searches, preference profile, location, and other attributes.

2.1.2 Main algorithmic approaches

We can generally divide them by their approach mentioned in [4] and [5] into:

- **Collaborative filtering** (CF)
  Solely based on feedback from users (user-item interactions). Trying to recommend unseen items liked by users with a similar taste based on other items they have both rated. And thus exploiting data of users with similar preferences.
- **Content-based filtering** (CB)
  Uses item features or descriptions to recommend items similar to those the user liked or interacted with. We are essentially building a model of preference for users and exploiting domain knowledge about items that match the users' model.
- **Constraint-based recommendation**
  It depends on hand-crafted deep knowledge about items. The user specifies a set of criteria based on which the system filters out items that meet the stated requirements. Additionally, the system can sort the items based on their properties if the stated criteria come with perceived importance - utility.
- **Hybrid systems**
  Combining multiple RS, either of the same type (with different parameters) or different types. This technique aims to increase recommendation efficacy. Main types according to [2] are:
  - Weighted where predictions of individual recommenders are summed up.
  - Mixed, where predictions of individual recommenders are combined into one list.
  - Cascade, where predictions of one recommender are fed as an input to another.

The popularity of the first two approaches varies from domain to domain. Some domains naturally contain item-specific data, which allows for the use of *content-based filtering*, for example, product parameters in e-shops. However, other domains do not. Then it is more beneficial to use *collaborative filtering* techniques or a mix of the two.

There are benefits and drawbacks to both. CF can extract latent meaning from the data that would remain inaccessible to CB that relies on items'

features. However, at the same time, it can cause problems to rely only on user-item interactions because we need a large amount of data to make a precise recommendation. There will be nothing to recommend if we cannot find similar enough other users that already rated some unseen items. This problem is called a *cold-start problem*.

The third technique, *Constraint-based filtering*, requires a deep knowledge that describes items on a higher level and is not very interesting due to the algorithmic simplicity. We will thus not discuss it further.

One other approach we did not include in the list is *Critique-based recommendation*. Its popularity is relatively low, but it is still worth mentioning. It acts as a guide through the item space, where in cycles, we show the user items that are distinct in some property (we could say they lie in different areas of the item space), and the user either accepts or rejects them. Based on this feedback(critique) from the user, we narrow down the user's preferences, offer different (filtered/extended) set/sets of items, and try to guide the user to a satisfactory result. The feedback can, in some cases be provided for not just the item but even specifically for its properties or part of the items. An example could be: 'This carpet has a beautiful pattern, but the color is not that nice'.

Some of the classical and more advanced methods include:

- User-based and item-based nearest neighbor similarity [6][7][8]
- Matrix Factorisation techniques[9]
- Deep Collaborative filtering [10][11][12]
- Deep Content extraction[12]

2.2 Group recommender systems

So far, we have discussed only recommender systems, where an object of a recommendation is a single user (from now on referred to as single-user RS or simply as RS, depending on the context). However, what do we do when we have a group of users we want to recommend to? For example, a group of friends selecting a movie they want to watch or a group of colleagues listening to music together?

Group recommender systems (group RS or GRS) are an interesting subarea of recommender systems, where the object of a recommendation is not just a single user but multiple individuals forming a group. The results of a recommendation for the group do need to reflect and balance individual preferences among all members.

2.2.1 Characteristics of group RS

There are many specifics that contrast GRS with single-user RS. Usually, some form of aggregation needs to be performed to transition from a single user preference that we gather to a recommendation for an entire group.

Situations differ for small and big groups, where the complexity of the users' preferences increases with the increasing number of users. At the same time,

every person is different, some may be more forgiving, and some may be less willing to conform to a different set of tastes than their own.

Although there are not many reported deployments of such systems yet, we see domains that would greatly benefit from using GRS.

Two domains, in particular, come to mind: movie recommendation and music recommendation. In recent years with the rise of popularity of streaming services such as Netflix and Spotify, there finally exists enough data and, more importantly, a proliferation of these services that would allow the utilization of GRS.

Group recommendation systems usually operate on top of single-user RS and then perform some aggregation in order to provide the user group with relevant recommendations. There are two main types of GRS. The first aggregates user preferences that are fed to a single-user recommender system. The second performs the aggregation on the output of a single-user RS and, in some way, aggregates the recommended items of each group member into a single list. We will talk in-depth about the possible methods and strategies in Chapter 4.

In our work, we focus on the second type that performs aggregation on top of an output of a single-user RS.

2.2.2 Challenges

We now mention some of the most critical challenges in the group recommendation domain.

- **How to merge individual preferences**
  The main problem when extending RS systems to support the group setting is how to combine individual users' preferences. It is possible not to support groups at all and let users deal with the act of combining them via discussion. However, the problem then collapses back to a single-user setting, where the user represents the whole group. Therefore, we need to decide how and when to merge them. The main two approaches are mentioned in Subsection 2.2.4.

- **Divergent group preferences**
  Some users are so-called *Grey-sheep* and *Black-sheep*. These users are hard to recommend to because their preferences do not align with many or any other users (respectively). This problem is especially hard to solve in Collaborative filtering, which directly relies on finding similarities between users. Furthermore, the same problem arises in the group setting, where it becomes much harder to find solutions that would be satisfactory to all of its members. So in Group RS, the problem of outlying users can be observed on two levels, in the usual situation, where the group aggregated preferences are outlined, and on another level, where the inter-group preferences of individual users do not match.

- **Feedback gathering**
  In most applications, feedback is gathered explicitly as well as implicitly. Explicitly by users rating recommended items and implicitly by the system observing users' behavior, such as which items they have visited or how

long they have interacted with the item. Gathering direct feedback in the group setting is still possible, even if it is more problematic due to the possibility that not all members leave a rating. In some cases, gathering indirect (implicit feedback) can become even impossible, depending on how the system-user interactions are designed. In most cases, users will be selecting an item on a single device under one person's account. Therefore, it is hard to distinguish between the preferences of that one individual and the preferences of the group.

- **Active/passive, primary/secondary group members**
  Another interesting issue arises when we consider that possibly not all members are equally important in the recommendation, as mentioned in [13]. One example could be when parents select a movie to watch with their children. The children should (arguably) be given priority over the selection. A second example could be that we would want to prioritize the satisfaction of individuals in the group who were less satisfied the last time an item was consumed.

- **How to explain provided recommendations**
  Explanation of single-user RS is already pretty challenging. With algorithms such as collaborative filtering, it is hard to explain why we are recommending an item apart from the obvious explanation that similar users liked this item. And with more advanced methods based on neural networks or more latent modeling of similarity and preference in general, it gets even more complex.

  This situation gets even worse, and another level of complexity emerges when we add the aggregation step to the recommendation process.

- **Not all members present**
  What can be done if we have a GRS preference model/data for a specific group and some of the members of that group are missing? We need to be able to modulate what members will be part of the recommendation process. This makes gathering and using feedback that would represent the whole group difficult.

  Another problem not entirely relevant to our work but still important is how we even know which group members are present. Many solutions exist, but they are not as seamless as the single-user variant, which is maybe one of the reasons why we do not see any widespread utilization of GRS so far.

- **Selecting from the provided list**
  Providing the group with recommendations is an algorithmic task, but we need to take the presentation and how the users operate the service into account too. If, for example, one specific group member is selecting an item for the whole group, let's say, a movie to watch, this member will most probably have the easiest way to propagate their feedback to the selection. Therefore we need to take the implicit feedback with a grain of salt and not consider it as an implicit feedback from the whole group.

  Further, graphical user interfaces and the setup of the whole service, how users interact and select items, leave feedback, and other factors become

important.

2.2.3 Classification

We now mention some basic classifications found in [14].

- **Individual preferences are known vs. developed over time**

  Some GRS start with a good knowledge of the preference of each group member, such as if we have a system on a popular music streaming service. On the other hand, some systems, such as expert recommender systems, can even start with no information about the group members. It then needs to develop and model these preferences over time using a critique approach.

- **Real time consumption vs. option presentation**

  There are two options for GRS recommendation depending on how the items are consumed.

  Firstly, we can provide a set of items that are further filtered by the group members before they select an item that they like. In this case, when the list is further narrowed down by the group members, we can recommend more controversial item options.

  Secondly, when the recommendations are immediately consumed, such as a music playlist at a social event. We miss the selection process with which the group members narrow down the items list. In this case, one good option is to present the users with a list of already liked/seen items. This playlist would be composed of items the users like on their own.

- **Group preference weight is identical vs. alterable**

  There are situations in which the priority of group members differs when it comes to satisfaction. We can have systems that allow setting different weights for individual members.

- **Type of recommendation output**

  We can have many types of outputs from the recommender system. Such as a single item, a list of a predefined length, a set that does not have an explicit order, and others. The output in the case of expert systems can be a graph, a set of rules, or a feedback-gathering question.

- **Single vs. k-item utility**

  Another difference is how are the resulting recommended items processed by the users as presented in [15],[16]. Do they select a single item, as is the case when recommending a list of movies? Do they consume the full list, such as playlist generation in the music domain? We must gather feedback and calculate the possible utility separately in these cases.

2.2.4 Common approaches

Now we introduce the two main algorithmic approaches of group recommender systems, according to [17] these are:

- **Group aware RS approach**
  Builds a model for the group based on all its members' preferences. Either directly by creating a model of preference for the group or by aggregating models of individual users together and then recommending items for the group as a single entity.

- **RS aggregation approach**
  Use single-user RS to recommend to each individual in the group and then aggregate the results together to create the final recommendation for the group.

  We can further split the aggregation approaches by where the aggregation takes place into the following two groups.

  - Aggregation of individuals' preferences before the recommendation and then performing recommendation as if the preference belonged to a single user.
  - Aggregation on top of recommendation results for individual users.

  We will further mention these two approaches in detail in Subsection 4.2.1.

In the RS aggregation approach, we further distinguish between two situations. A situation where we have predictions for all possible items and, therefore, can aggregate directly on the ratings of all items. And a situation where we only have a list of items (a subset of all items) for each user. These two can function very differently, for example, taking in context only the position in the recommended list or position and the rating. They are mentioned separately in [17]. From a different point of view, they only differ in the availability of provided results from the underlying RS, so we group them under one main direction.

Further, both group-aware RS and aggregation approaches have advantages and disadvantages. One of the advantages of the Aggregation approach is that we can use the same RS as we would use for an individual recommendation. Either as a black box, directly performing aggregation on the top items that the black box provided, or in a more involved way by utilizing the predicted ratings. However, the aggregation strategies do rely on single-user RS so there is not much that can be done in order to extract some hidden latent preferences of the group, which in the case of the first method, the group-aware approach, can potentially be extracted.

We will discuss techniques from the latest literature in-depth in Chapter 4.

At the same time, we need to define what it even means to recommend something to a group. Do we measure it by fairness, overall user satisfaction, or by the least satisfied member of the group? We will describe common approaches to these problems in Chapter 3.

3. Fairness

So far, we have discussed recommender systems in general and the methods that are used in the field. However, now, let us step back and look at the problem from a broader perspective of fairness as a social construct. Specifically, we will focus on the importance of fairness in the context of algorithms. What role fairness plays when using machine learning models with potentially sensitive data, and if we can make group recommenders better when we understand and define fairness and its underlying properties better.

We will start with a general introduction to the topic of fairness, define its possible meanings and specify which one is important in our setting. This is required due to the overload of the word itself and the rising importance of the topic in today's world. Furthermore, we will explain why fairness seems to be a crucial parameter in the group recommender setting and will try to reason about how to measure its effects.

sectionGeneral

The word fairness itself is hard and controversial to define. In Cambridge Dictionary [18], it is defined as "The quality of treating people equally or in a way that is reasonable." Its use has been rising steadily since the 1960s, as we can see in Figure 3.1.

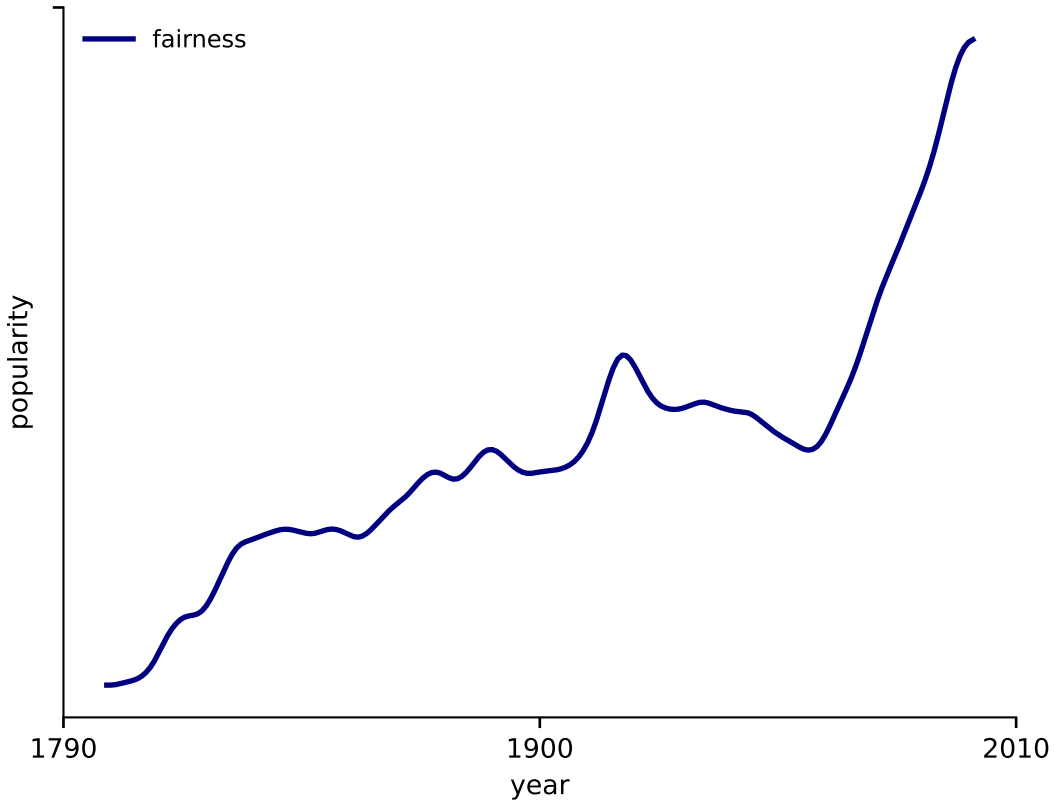


Figure 3.1: The graph shows the phrase "fairness" occurring in the corpus of English books from 1800 to 2010. Source: Google Ngram Viewer, corpora 2012. [19]

Humans are obsessed with fairness. From a young age, children will get sad

when something is not fair, when their sibling gets a more significant piece of the pie, more attention from their parent, or any other unequal situation. It can induce strong emotions such as envy, sadness, or anger, and it emerges very early, as early as 12 months of age or sooner, as researched in [20].

Furthermore, this behavior is not limited only to humans. We can observe the same behavior in monkeys. In [21], the authors observed that if a monkey is getting the worse reward for the same task as its peer, it refuses the reward and demands the same payout even if they were satisfied with the lesser reward for the same task before. In some sense, this is very strange; why should we care about someone having more if we have enough?

We observe the same behavior in many other species of animals, but not all. It seems that it requires a certain level of intelligence for the notion of fairness to emerge. As discussed in [22], based on studies of other non-human species, this evolutionary puzzle can be further dissected into responses to reward distribution among the cooperating group members. Humans are willing to seek an equalization of outcomes even if it means that they will lose some of their own reward as studied in [23]. At the same time, we humans like 'free rides' where we get high rewards for a small amount of work but dislike when someone else gets the same. This directly corresponds with fairness itself — It is not fair if someone else gets something we do not. However, it all depends on our personality and the type of relationship involved. Some people are, for example, willing to accept that their partner makes more, but that is very subjective and sometimes can even lead to larger envy.

In some cases, we observe a pattern of generosity, where children are willing to make their own sacrifices in order to ensure fairness so that the other person does not have less than them, as presented in [24]. The authors studied fairness in multiple cultural settings and found that this behavior is learned and only present in some cultures.

On the other hand, our society widely accepts the notion of 'winner takes all', which can be problematic in itself, for example, in sports and business. In business, it directly shapes the distribution of wealth, which is considered one of the main problems of today's world. Nevertheless, discussing the social aspect of fairness is beyond the scope of this thesis.

Regarding the true nature of fairness on the deepest level, it could even be possible that the notion of fairness emerges together with cooperation, language, and communication. It would mean that it is an inherent property of any intelligent agent created through an evolutionary process. Another point of view could be that fairness acts as a mechanism that pushes towards equality among the group members, leading to higher stability of the group, which would give an evolutionary advantage.

However, more research has to be done, as our understanding of intelligence, consciousness, and related hard-to-quantify phenomena are lacking.

Let us now get back to fairness in the context of computer systems.

3.1 Algorithmic fairness

We will focus on fairness regarding society or individuals interacting with a computer system. We will not discuss further any meaning of fairness outside of the

domain of computer science. This topic steers away from the primary goal of defining fairness for the group RS sub-domain, but we think that it is a very important topic in general, and therefore it deserves more spotlight and can be an excellent middle step to understanding specifics of the group RS setting.

3.1.1 Sources of algorithmic unfairness

How can a computer with no underlying understanding of race or ethnicity discriminate against a group of people? At first, this idea may seem strange, but we have to remember that, as with any other computer program, machine learning algorithms are designed by people. Data used to train those algorithms comes from the real world, where bias and unfairness are unfortunately still present. Thus, the trained models, even if not usually meant with an ill-fated purpose, will reflect that and, in most cases, include some form of bias or unfairness. Of course, there exist uses of ML with bias that has been introduced on purpose. We can see that for example in the Chinese credit score system which is biased towards an individual based on their class, race, political views, and other factors. Which we, in the modern democratic world, consider as protected(sensitive) characteristics, that are not to be used to discriminate against an individual. However, this type of bias in those circumstances is a knowingly designed and required feature of the system; therefore, we will not discuss these systems any further.

More rigorously, we can say that output of any machine learning (ML) algorithm is usually just a product of the underlying data. An accurate classifier will reproduce patterns found in the training data by design. Usually, the bias is either transferred directly from the data or by wrongly defining the learning objective.

Let us now present a division by the main sources of unfairness as stated in [25] with examples following in Subsection 3.1.2:

- **Biases already included in the dataset**
  Such as dependence/correlation of data based on sensitive characteristics. An accurate classifier by design reproduces bias found in the data.

- **Biases caused by missing data**
  Missing or filtering out some of the training data can result in a dataset that does not represent the target population.

- **Biases stemming from algorithmic objectives**
  While training, we usually minimize some error, which can lead to a prioritization of the majority's interests if left unchecked. It will always be easier to optimize results for groups with small entropy, than for niche groups that are more surprising and thus have a larger entropy.

- **Biases caused by "proxy" attributes**
  Some attributes that are not directly considered sensitive can still contain information from sensitive attributes. In other words, they are not independent. Therefore, the algorithm can use the "proxy" attribute and indirectly exploit the sensitive attribute.

It is important to define which sensitive characteristics need to be considered. As stated in [26] those are gender, ethnicity, sexual orientation, disability, and

others. Most research in the domain of bias and fairness is, based on our perception, studied from the perspective of discrimination and the impacts of algorithms on society. We observe bias against specific groups of the population based on their race, sex, nationality, education, beliefs, and many other attributes (protected as well as unprotected ones), which causes a measurable impact on our everyday life. After all, we are more than ever involved and surrounded by technology. Therefore, it is essential to understand the effects biased algorithms can have and to study techniques and strategies to mitigate their negative impact.

Further, we can also view fairness from the aspect of algorithmic decision-making, where a decision process can introduce unfairness based on some non-deterministic property or computation. Some sectors, such as justice and finance, have to strive for equality of outcome due to the high cost of errors of unfair decisions, either in the form of unjust punishments in the former case or financial loss in the latter.

3.1.2 Examples of algorithmic bias

We will now present a few instances of computer systems that have been used in settings where a bias towards a sensitive characteristic had a substantial impact.

- **Amazon's automatic recruiting system**
  As reported by [27], in 2018, it was found that the new system for hiring people for Amazon was biased toward women. The IT field is mostly male-dominated - women represent only around 23% as stated in [28]. Due to this disproportion, the algorithm discovered in training data pattern between the gender of the candidate and hiring results which led it to assume that male candidates are preferred before female ones. The algorithm was not told the gender of the candidate, but it inferred it from other data such as university, hobbies, and others. This bias is a combination of 'bias already included in the dataset' and 'biases caused by proxy attributes'. The company later disbanded the team and left the tool only as a helper tool that works in conjunction with the recruiters instead of solely automatically.

- **Apple's credit card**
  Apple released its credit card in 2019. It works as follows: after the sign-up, the user receives a certain credit limit from the service provider (Goldman Sachs). Some people, as reported in [29], noticed that their wives were assigned smaller credit limits even though their credit score was higher and they only had one shared bank account. In this case, an investigation by the New York State Department of Financial Services came to a conclusion based on an extensive analysis that no unlawful discrimination against applicants has taken place.

- **COMPAS** - Correctional Offender Management Profiling for Alternative Sanctions
  COMPAS is an algorithm used in the US justice system to predict the likelihood of a defendant becoming a recidivist. Analysis [30] found that black defendants were often predicted as being at higher risk than they actually were. On the contrary, white defendants were predicted to be

less risky than in reality. In the case of re-offended, this predicted risk was almost twice as high for blacks compared to whites. They conclude that the tool is imprecise and does not reflect the actual likelihood it was designed to predict.

In these cases, society, and mainly the law, has to act and protect those that are treated unfairly. European law can act as an excellent example of what can be done. The general data protection regulation (GDPR) and the protection of individuals against algorithmic bias are great and functioning examples.

Details about laws that are in effect in the EU and definitions of sensitive characteristics and areas of protection can be found in Handbook on European non-discrimination law [26].

3.1.3 Measures of algorithmic fairness

We need to have a precise way of measuring bias towards sensitive characteristics to design and evaluate algorithms that are taking or should take measures to ensure fairness.

At first sight, our idea could be to remove features that we consider sensitive entirely from the dataset, but that will, in most cases, not suffice due to other features being slightly correlated with the sensitive feature. Correlation with, for example, gender will probably be too small to be predicted with measurable accuracy. However, this balance can tip over if we combine many of these slightly correlated features. We, therefore, need to approach this problem more rigorously.

We now present a few statistical methods as mentioned in [31]:

- **Independence** We say that sensitive characteristic Char is independent of a prediction Pred if:

$$P\left(Pred = p | Ch = a\right) = P\left(Pred = p | Char = b\right) \quad \forall p \in Pred \quad \forall a, b \in S \tag{3.1}$$

  is the probability of the given prediction being the same for two people from different groups with respect to the sensitive characteristic.

- **Separation** We say that random variables $(Pred, Char, Y)$ satisfy separation if the sensitive characteristics Ch are statistically independent of the target value Y given the prediction $Pred$. This relation can be expressed with:

$$\begin{aligned} P\left(Pred = p | Y = q, Char = a\right) = P\left(Pred = p | Y = a, Char = b\right) \\ \forall p \in Pred \quad q \in Y \quad \forall a, b \in Char \end{aligned} \tag{3.2}$$

  Meaning that the dependence of a prediction result on the sensitive attribute $Char$ can be justified by the attribute $Char$ being dependent on Y.

- **Sufficiency** We say the random variables $(Pred, Char, Y)$ satisfy sufficiency if the sensitive characteristics A are statistically independent of the target value Y given the prediction R. This can be expressed as:

$$\begin{aligned} P\left(Y = q | Pred = p, Char = a\right) = P\left(Y = q | Pred = p, Char = b\right) \\ \forall q \in Y \quad p \in Pred \quad \forall a, b \in Char \end{aligned} \tag{3.3}$$

  We say that $Pred$ satisfies sufficiency i the target variable Y and the sensitive attribute $Char$ are clear from the context.

3.1.4 Outcome vs. opportunity

From separation and sufficiency, we see that there is only a difference in the direction of the relationship between random variables $Pred$ and Y. We will call them 'equality of opportunity' and 'equality of outcome', respectively.

Let us now present an example of both. We have a model situation where we strive for gender equality in the management of our company.

- **Equality of outcome** We would like the resulting distribution to be fair such as there has to be 50% male and 50% female gender representation among our management. If we have more than 50% men, we need to fire them and hire only women. It may seem easy, and this is where most of the efforts usually stop, but even just finding what the desired resulting distribution needs to look like is a non-trivial task.

- **Equality of opportunity** In this case, we try to mitigate any bias that could skew the decision of whom to hire towards any gender. Preferably, we do not want to even propagate the fact about the protected attribute (in this case, gender) to the people making the final decision.

Both of these cases/methods have their place but should be used cautiously. They can cause a great deal of fairness equalization when used correctly but, at the same time a great deal of harm when implemented incorrectly.

With the already discussed topics in mind, we can connect the fairness and the group recommendation systems with the subsequent possible interpretation. What applies to our group recommender domain is the notion of fairness in the sense of a balance of preference between group members. Each member has their preference, and we are trying to balance them in the best possible way so that everyone likes the recommended object or list of objects equally.

Further, if we take group membership as a sensitive attribute of the group and consider it a sensitive attribute, then we want independence in the context of equality of outcome.

3.1.5 General methods of prevention

Thanks to machine learning models being entirely dependent on the data, as discussed previously, we can divide the general techniques of bias suppression by where the change to the machine learning process is made on the data path. We divide the general techniques into three categories as follows:

- **Pre-processing** Adjust training data in a way that sensitive characteristics are uncorrelated.

  The main benefit of preprocessing data this way is that if we ensure independence this way, then any subsequent deterministic method will transitively also satisfy the independence of the sensitive attributes. Nevertheless, as with any data transformation that changes data properties and distributions, we need to be careful not to hinder the efficacy of the final model.

- **At training time** Design algorithms that set constraints on the optimization process itself.

This change seems to be the hardest to technically and algorithmically implement. We need access to the whole collection of raw data to ensure that we are not biased. Furthermore, we limit ourselves to ML methods that allow this constrain modulation. On the other hand, the main benefit is that we perform the optimization with full information of the task and, therefore, can potentially gain more utility.

- **Post-processing** Adjust parameters of the already learned classifier so as to be uncorrelated with the sensitive attribute.

  This is the least favorable from the theoretical point of view due to us only being able to correct the bias in the way of ensuring equality of outcome, which corresponds with the difference between the before-mentioned properties of sufficiency and separation. At the same time, it makes the evaluation of the model performance harder.

3.1.6 Other adverse effects

It takes a lot of time to make datasets and models unbiased, especially because bias is included in most of the datasets that come from the real world. Let us now discuss some other machine learning effects that play a big role when combating algorithmic fairness. We now need to take into account the iterative learning of the machine learning models, meaning that we, after putting them to production, retrain the model on data that was affected or directly generated in or by an environment that the models were part of.

Two of the most common problems are the so-called 'negative feedback loop' and 'echo chambers'.

Negative feedback loop

A decision-making system that is learning from past data and is retrained in the future on data that were affected by its decisions (after being utilized in the environment from which we gather the dataset) will consume its own (previous versions') decisions as training data. This can lead to the amplification of biases that were already present and to further skew the distribution of the underlying data. This effect is called a 'negative feedback loop'.

To build robust and fair algorithmic systems, we must understand when and why this effect emerges. When we look at it from a simpler perspective, where the current system deployed in production is just a set of predictions, then training the next model is just training the previous with additional data that the current model made (the set of predictions from production). In this sense, the set of features selected while the current model was trained will correlate with the new data and therefore affect the selected and used features in the new training. This can be very detrimental to the performance of the model. And it is not easily fixable due to how data is gathered and how machine learning is iterated.

We will mention an example from [32]. Let us assume that we are building a decision-making algorithm that decides where to put our call-center capacity in order to generate the most profit. In other words, to which telephone numbers from some candidate list to call. And let's again assume that in the prior data, the conversion rate of a person coming from page X is 5%, from page Y is 2%, and

from page, Z is 1%. The algorithm will naturally be selecting to focus our capacity to X, because it has the biggest conversion rate. Now fast forward some time when we are retraining the model on new data. Due to us preferring page X, and not calling people coming from Y and Z, our data distribution of conversion rate looks like this: X: 8.5%, Y: 0.5%, and Z: 0%. We end up having less data about Y and Z due to us not calling people coming from those pages. Now we will retrain the model, and this bias towards X will reinforce. In this way, the performance of the model is decreasing due to the fact that data no longer following the actual underlying distribution. If we use concepts from Reinforcement learning - we are exploiting but not exploring.

This negative feedback loop can be and is very detrimental to models' performance and, in a wider view, even dangerous to society. This directly leads us to a second adverse effect of echo chambers.

Echo chambers

The feedback loop described in the previous section can be observed in effect not only when retraining algorithms but in social groups as well. People naturally seek out information that reinforces and supports their existing views, as stated in [33]. Serving content to users that support their views will therefore lead to higher satisfaction and interaction with the system.

The main problem becomes the training metrics that try to maximize numerous aspects of the system's performance, such as the amount of time users spend interacting with the service and the return rate of the visitors. We use them as a target for optimization, and that leads to the creation of systems that naturally lock its users in so-called 'echo chambers'.

While being inside, your own views will be reinforced. This, together with people increasingly consuming social media feed as their main source of news, is probably one of the main forces behind the increasing polarization in society, as discussed in [34] and [35].

3.2 Fairness in Group recommender systems

So far, we have discussed algorithmic fairness in the context of equality of opportunity, where the main goal is not to discriminate against an individual. The same issues are present in the Group RS domain, but we will focus more on equality of outcome. Our objective is to fairly distribute item recommendation quality between group members so that everyone is as satisfied as possible and the level of satisfaction among users is as close to uniform as possible.

Let us first introduce two concepts of item preference:

- **Member-likeable**
  We say that a recommended item is member-likable if it is chosen with the aim of satisfying a single-group member or a small subset of a group more than with the aim of increasing the average.

- **Group-likeable**
  We say that an item is group-likable if it is selected for recommendation with the goal of uniformly satisfying all group members.

This is one of the main optimization decisions we have to make. In some cases, as we will discuss later, member-likeability will be a better choice. In other cases, it will be the group-likeability. We can view it as an opposing force. We can either push towards more or less uniformity.

3.2.1 Specific cases of fairness

Let us now mention some of the main ways and their differences as to how fairness in this context can be understood.

- **Fairness distribution in isolated recommendation**
  We recommend a list of items (possibly even a single item) and have to balance the choice of the items so that all members will be satisfied. This single list is an isolated recommendation. We can view fairness in this setting as a direct optimization problem, where items are considered better if they are liked on average (among the group members) more than items liked by some part of the group and disliked by the other. As the group size grows, this is harder and harder to satisfy because a larger group will most probably have a broader preference which will be harder to meet due to the differences in members' tastes. We can view the preference of a group more like a set intersection than a set union.

- **Fairness in a list of items that are consumed sequentially** This setting differs from the previous one because we can recommend items that are less universally likable but more specific and only liked by a part of the group. Therefore intertwining the items so that each member will be satisfied "at some point". The balance of group-likable or member-likable items can be tuned according to our specific requirements.

- **Long-term fairness**
  Further, we can distinguish another case where recommendations are provided in batches separated by some more significant amount of time (days or longer). This case is somewhat similar to the last one but differs in the fact that unfairness can be more costly to repair. If a person dislikes the item recommended by a group RS, they will less likely to be part of the recommendation in the future. So the balance mentioned in the previous setting is an even more important and sensitive parameter to tune. And at the same time, it is more important to gather and process feedback.

- **Uneven importance of the group members** In some cases, there will be a situation where the expectation of fairness is distributed non-uniformly. For example, when watching a movie with your kids, you probably care about the satisfaction of your kids more than your own. But at the same time, you want to take yourself into account too. In these cases, it is essential to view required fairness as a fluid parameter that can be modified and satisfied by uneven criteria towards group members.

3.2.2 Evaluation

In order to assess the performance of group recommender systems, we need to define some metrics as to how to measure the fairness and other utility function we select as the observed parameter. Let's assume that an output of a group recommender system is a list of items recommended to the group of users. For each item, we have information about the effect of that item on the user's utility function.

The utility function we want to calculate is the total relevance of the recommended list to each user, and we consider it fair if it is balanced among users, where no one is systematically biased against. We, therefore, need to provide a single measure for each user and the recommended item list.

- **Average relevance score** (AR) We calculate a simple average of the observed metric for each user separately, such as the expected relevance of the item to the user. We can then evaluate for each group minimum, maximum, average (and other) of this average aggregated relevance. We, therefore, end up with a set of evaluations for each group, which we can further process.

- **Normalized discounted cumulative gain** (nDCG) We can assume that the position of items in the recommended list is important. Especially with fairness. As an example, if we generate a playlist for a group of two users and then satisfy the first user for the first half, the other user will be quite unhappy even if the second part of the playlist will be dedicated to them entirely. This approach would not be ideal. It is, therefore, important to incorporate the position of the items accordingly.

  Discounted cumulative gain (DCG) is defined as:

  $$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)} \tag{3.4}$$

  .

  And further, nDCG is a normalized form where we order items in the list p by their relevance so that they create the ideal list with the maximum DCG score. We assign this score as $IDCG_p$ (ideal-DCG).

  $$nDCG_p = \frac{DCF_p}{IDCG_p} \tag{3.5}$$

  .

With these two measures of recommendation, we have a way how to aggregate the relevance of items in the recommended list. We will mention further specifics about evaluation in Chapter 7.

3.3 Other criteria

We have discussed fairness as one of the criteria for the evaluation and optimization of RS and ML tasks, understandably so, as it is the main topic of this thesis. But what for is an algorithm that is as fair as possible if its outputs are

disliked? We need to have an overview of other optimization criteria in order to design algorithms that will be useful in real-world applications. Fairness, as well as most other parameters we are usually trying to optimize is a part of a general multi-criterion optimization which we call - the real world. Let us now introduce some of the most popular criteria that we can aim to optimize.

Bias

As already mentioned before, bias is an important harmful property that we are aiming to reduce. Examples of bias are all around us. Each person has at least some prejudice and false ideas about topics in which they have no expertise. It is, therefore, important that we actively try to broaden our minds and actively suppress these biases. Just then, we can truly become equal as individuals.

From the algorithmic view, bias is mostly introduced from the underlying data that we are using as a training set. Sometimes these biases are even knowingly used to generate profit. For example, in marketing, where sex is a very good indicator of preference. A good question arises - where lies the line between fair usage and abuse? We do not know yet, but it seems that privacy and fairness will be gaining importance, and therefore the question of how to get rid of bias will become more significant.

Privacy

We have to extend our definition of privacy to ML systems as well. It tries to model given underlying data and, therefore, can leak users' information that is present in the data if the model is 'copying' the underlying dataset too well. Some models are explicitly based on the similarity between users. A great example can be user-based collaborative methods from recommenders systems that were described in Subsection 2.1.2. In collaborative methods, we first find users that are similar to our target user to whom we are generating a recommendation, and then we recommend those items that similar users liked and our user have not yet seen. It can inherently happen that this preference we are unknowingly exploiting can be considered private. In this approach lies the great strength of user-based collaborative methods, they can extract latent meaning from data that would stay otherwise hidden. But on the other hand, it can lead to a breach of privacy. Another great example can be text processing systems that are taught on a huge corpus of diverse text ranging from books, private messages, code repositories, and other sources. A good example is a coding assistant for generating suggestions called GitHub Copilot. After the release of an initial version, it was discovered that it leaks secret information from the code repositories that it was trained on, as discussed in [36] and [37], more specifically - API access keys. The issue was later mitigated by applying content filters that check for known secret data such as the already mentioned API access keys, emails, passwords, and other personal data.

Other issues apart from private data propagation out of training datasets to predictions of the models can be the gathering and usage of massive training datasets themselves. Fair use, which is vaguely mentioned by almost all service providers that gather users' data, should be revised and brought up to the

standards of the 21st century.

Another dangerous privacy breach can be the identification of users from published datasets. We need to protect the gathered data and anatomize them so that the potentially private data cannot be traced back to the actual people. One major privacy leak happened with the second Netflix challenge, as mentioned in [38] using methods from [39]. With only a very small amount of a person's movie watch history, for example, extracted from a public source such as IMDb.com, we can match the data from the Netflix challenge to this public source. The Netflix challenge was later canceled.

Trust

Different machine learning applications usually require a very contrasting emphasis on optimizing trust. It directly corresponds with how high of an impact the provided algorithmic decision can have. We will put a system that recommends books to a way lower level of scrutiny than a system that recommends which medication or treatment a person should get. This direct proportion between the importance of machine learning output and a potential personal loss needs to be taken into account while designing the system. Sometimes even a highly effective and correct output of a machine learning application can be dismissed only due to a lack of trust in the application. Then, the effectiveness of the system can diminish even if the training and testing data tell otherwise. Trust can be greatly increased when providing explanations accompanying the output of the system.

Explainability

Having a reasonable explanation as to why we are getting the result we are getting can greatly improve the real-world effectiveness of the system. It can decrease negative reactions to non-ideal predictions and make users more forgivable, as stated in [40]. We provide two examples of how can such an explanation look like, first from the Amazon book store in Figure 3.2 and second from Facebook's explanation of why a particular advertisement served in Figure 3.3.

Explanations can achieve not only increased trust in the system, as mentioned before, but an increase in transparency, scrutability, persuasiveness, overall satisfaction, and more. The main problem is, that explanations are generally hard to provide, and they need to be taken into account while developing machine learning systems in all stages. Some algorithms are currently very hard or outright impossible to explain. One of the proliferated examples is neural networks. Neural nets keep the context in neural connections, which are very hard to provide an explanation for. We sometimes refer to this property of a system we do not fully understand as a 'black-box' system. We know how they work and how to train them, but due to the inner complexity, we fail to explain the exact way how the result was calculated. That leads to methods that approximate the black-box behavior and try to provide explanations with some varying degree of inaccuracy, such as [41].

Explainability can even be indispensable. Let us assume that an ML model is used to recommend legal action against an individual in a judiciary setting. We cannot soundly present it as evidence without being able to justify its actions

and warrant its correctness.

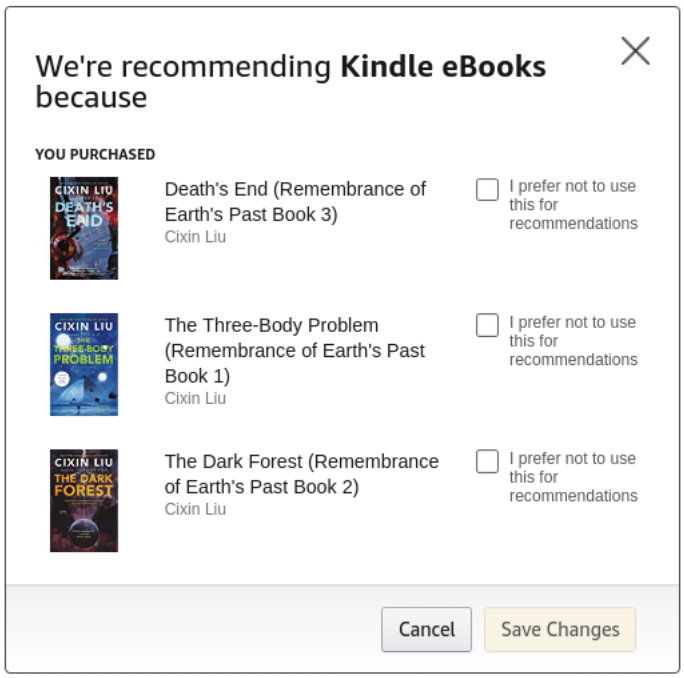


Figure 3.2: Amazon's web-page window that provides context why was a particular book recommended.

Legitimacy

Legitimacy can be viewed as another step after explainability. When we provide a prediction together with sound reasoning that can be verified either by a human analyst or by another trusted system, then we can use it in more serious and sensitive deployments. One of them can be the legal domain. In it, computer systems can be used to provide unbiased and fair decisions and analyses if set up correctly. But as mentioned many times before, we need to be very careful with the data we use for training. Care needs to be taken even if we do not use an automated training solution. Most of the current systems used in the mentioned legal domain and other systems where legitimacy is required are most probably based on hand-crafted rules that can very well be subject to bias as well. One of the examples is the system COMPAS presented in Subsection 3.1.2.

Consistency

For some systems, it is very important to stay consistent in outputs even with slightly different inputs. Let us assume that we are developing an illness detection algorithm that offers medical personnel some insights about their patients based on provided symptoms. The predicted recommendation should not drastically change if, for example, the temperature changes by 0.1°C. It is, therefore, closely tied to the previous properties of legitimacy and explainability. Most high-critical ML systems have to be designed with consistency in mind. Driverless cars that abruptly change direction with only a one-pixel change in the input would not induce the trust of their users.

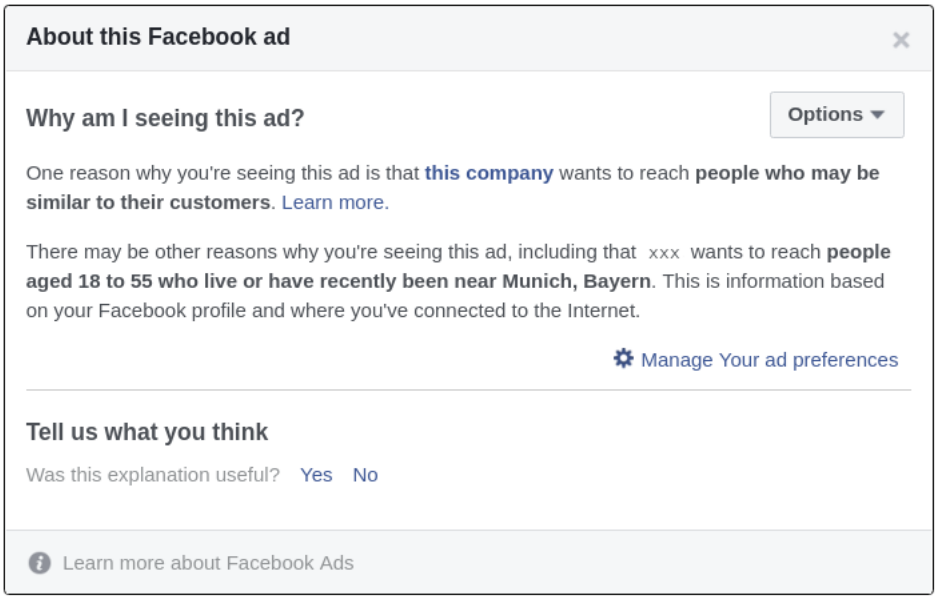


Figure 3.3: Facebook's web-page window that provides information about what data led to users seeing Facebook's ads.

From a different perspective, we can view fairness as a sort of consistency in the way that protected attributes of people do not affect the output. The algorithm is consistent in respect of these protected attributes.

One other meaning of consistency in regards to ML systems can be stability in the meaning of ML outputs being the same while navigating a system or an app. We can attribute it to the application's design more than the ML system itself. If a user is browsing a web page, let us say an aggregator of products on the web, they may rely on the back button while browsing the different outputs. In this setting, inconsistency and changes to the recommendation items may be harmful as they will affect the user experience while listing and using the provided information.

Novelty

Novelty can have multiple meanings when we use it to describe RS. Firstly it can be that the item itself is a new addition to the dataset and we do not have many user-item interactions which we would use to asses and recommend the new item.

Secondly, we can say that the item is presented to the user for the first time. Sometimes, depending on the presentation of the recommended items, we have to show the recommended item repeatedly in order for it to be noticed. In that way, novelty can be viewed as a non-binary attribute, where each time we present the item to the user, it decreases the item's novelty with respect to that one user. But often, it is viewed as a binary attribute, shown or not-yet-shown.

Thirdly, by novelty, we can describe an unusual item presented to the user. Such a situation can occur either due to the RS incorrectly assuming that the user will like that item or, on purpose, introducing a new, not yet seen item so that we introduce more exploration and present items that the user will view as fresh.

And lastly, novelty is important as an exploration. RS should try to broaden

and expand the model of users' preferences, which needs to be done by gathering feedback on items outside of the users' current preference model. A balance between exploration and exploitation is difficult to tune, but it can substantially increase models' performance if done correctly.

In general and contrary to the previously mentioned consistency parameter, we usually strive for novelty and exploration. If a person is picking out a movie to watch, then showing them the same selection over and over would most certainly lead to their dissatisfaction due to the limited and repeated content that the system is providing. As with all properties, novelty requirements heavily depend on the domain the system is deployed in. This property is most sought for in domains where exploration is important.

Contrary to the previously mentioned consistency parameter, we sometimes strive for novelty and exploration. If a person is picking out a movie to watch, then showing them the same selection over and over would most certainly lead to their dissatisfaction due to the limited and repeated content that the system is providing. This property is most sought after in domains where exploration is important.

Coverage

In some ML and RS applications, there can emerge a situation where some items are never recommended. If the reason is that the item quality is bad then it can be the right system quality. In other situations, such as RS that recommends based on item properties, an undesirable behavior can occur when an item is just too different, and we do not have a reasonable 'link' to other items that would serve as a comparison for the recommendation. Then this item will be left out and never recommended.

Another coverage problem is with popular items. Popularity is sometimes a good indicator of quality, but not always. If an item is popular, then in a system that exploits popularity, it will receive more and more attention. This, in turn, further increases its popularity. In a way, we can have the same type of phenomenon as the negative feedback loop mentioned in Subsection 3.1.6. It then happens that item exposure distribution will be unnaturally skewed even more towards the popular items. This can lead to a decrease in the system's performance due to some less popular items not being recommended, even if possibly a better choice for a recommendation. In a way, popularity and unpopularity correspond to a notion of novelty where instead of considering a single user, we consider all users of the system together.

Efficacy, accuracy, and precision

We have so far mentioned multiple other criteria. But all of them are hard to measure due to their somewhat subjective nature. Let us now present two of the most widely used exact efficacy properties: accuracy and fairness.

Accuracy (sometimes trueness) is a measure of how far away the outputs of an ML system are from the desired outputs. Precision is a measure of how unsure or dispersed the outputs are, in other words, how away are from each other. They are both measures of observation error, and the definitions differ based on the

type of the outputs of the system. We can see an illustration for continuous value prediction on Figure 3.4

Figure 3.4: Illustrative description of the relationship of accuracy and precision from [42]

How to measure the distance of outputs to ground truth can differ based on the shape/dimensionality of the output. Usually, we use a simple metric such as Euclidean distance or simple Manhattan distance.

For categorical data, we have (binary retrieval task and multi-class classification, respectively):

$$Accuracy = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} = \frac{\text{correct classifications}}{\text{all classifications}} \tag{3.6}$$

and precision for binary classification:

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.7}$$

Precision for categorical data does not have a set meaning, we can either use precision for each class separately or have one class that acts as an FP label.

Difficulties with using accuracy arise when labels are missing some or all items in the dataset. Calculating the correct denominator in categorical cases or the correct distance to the reference value can be difficult or/and impossible.

Performance and scalability

We can design any algorithm we want, but what for will it be if we cannot then use it in the real world? Theoretical algorithms certainly have their place, but we are aiming to solve real problems of addressing fairness. Therefore another and last discussed property will be performed and, with it, the related property of scalability.

The performance of a system can be described either in a form of total computational resources needed for processing, or the whole deployment or normalized to a single user. Another possible performance property can be the length of time needed for the processing of a single request called latency. These two properties usually need to be balanced against each other. To a certain point, we can either add more computational capacity to improve - decrease latency, or remove some capacity with the possible increase in latency.

Scalability is a property of a system that it is able to handle a growing amount of work. We have to keep this property in mind while designing ML and RS systems. If, for example, we are using a comparison with other users, such as in the case of some RS algorithm, we will be growing the system requirements most probably linearly with the number of users. This can be fine in for example research applications but becomes a real concern when applied to worldwide applications used by millions of users. At these scales, we will be required to scale using not only vertical scaling (that is using more powerful computational resources), but mainly using horizontal scaling (by using many computers at once).

4. Related work

In this chapter, we will go in-depth discussing common approaches that are being used in the group recommendation task. Firstly, we specify the main approaches and divide them into groups by how they approach the transition from single-user preferences to group results. Then we mention the main simple techniques and describe how they interact with the usual recommendation approaches. Lastly, we dive into more advanced methods that perform the aggregation more elaborately.

4.1 Categories

We assume that we have individual user preferences (if group preferences are available, then the task becomes a simple recommendation with the group acting as a single user); therefore, it is necessary to make a distinction based on where the algorithm goes from the preference of the group's users to a result for the whole group.

We can put each algorithm into one of the following three groups based on the aggregation step:

- **Aggregate models**
  The aggregation works on merging the preferences of each group member into a single set of preferences that a recommender system can directly consume, therefore creating a group preference model. Aggregating the single-user preferences either directly by aggregating ratings of seen, or rated items, or by aggregating the extracted models of user preference to create a single model for the whole group, such as preference matrix in matrix factorization approaches, text descriptions, or item-based recommendations and so on, we will discuss these techniques later.

  This aggregation step precedes the recommendation step. We can see a visualization in Figure 4.1.

- **Aggregate predictions**
  Aggregates predictions outputted from RS. Recommender recommends separately for each group member based solely on their single-user preferences. Then the resulting recommended items are aggregated into a single list of recommendations for the whole group. There are two main ways how the final list can be created. Either directly take items recommended to each user and append them together in some specified manner, or the second way, calculate some utility function from all recommended items and select those most fitting to the group based on this utility function. We will discuss both in more detail in Section 4.2.

  The aggregation step follows after the actual recommendation step. We can see a visualization of this approach in Figure 4.2.

- **Aggregation is a uniform part of the recommender**
  In this case, the algorithm directly works with group users and does not allow for a clear distinction of the aggregation step. It is deeply and inseparably built into the algorithm itself. Sometimes the perception of the

inseparability of these two steps can vary in the literature. For example, aggregating the user profiles in matrix factorization makes the aggregation inseparable because there is a specific preprocessing done before the aggregation step. However, others will point out that the user latent matrix is just a representation of a user preference, even if processed by the algorithm itself. We will let the reader decide where they see the distinguishing border. We will briefly discuss the available methods in 4.3

This presented grouping based on the aggregation step is different from the related literature [17], and [2], where they also make a distinction into three groups but based on the data that the aggregation processes and the position. Instead, we have chosen a little different distinction based more on the interaction of the aggregation with the recommender system, essentially putting two groups from the literature mentioned above under a single group (*aggregate models*), but with the mentioned possible subdivision.

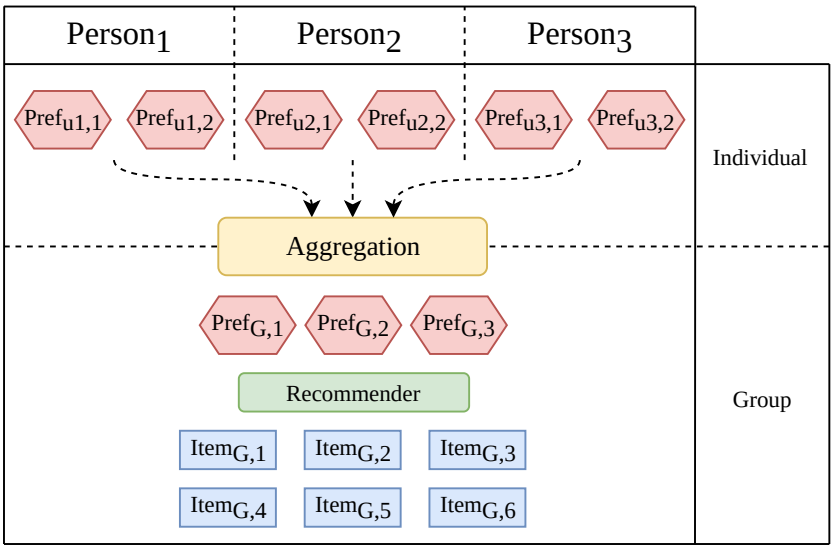


Figure 4.1: High-level overview of group recommendation with aggregation of individuals' preferences, before recommendation.

4.2 Simple aggregation methods

We will now introduce the main aggregation functions (interchangeably as 'aggregation strategies', 'aggregation methods') used, together with an overview of how they interact with the single-user recommender systems introduced in Subsection 2.1.2.

4.2.1 Methods

Aggregation methods can be divided into three groups based on their high-level approach: majority-based, consensus-based, and borderline-based strategies. The majority-based generally uses the most popular item among the group members,

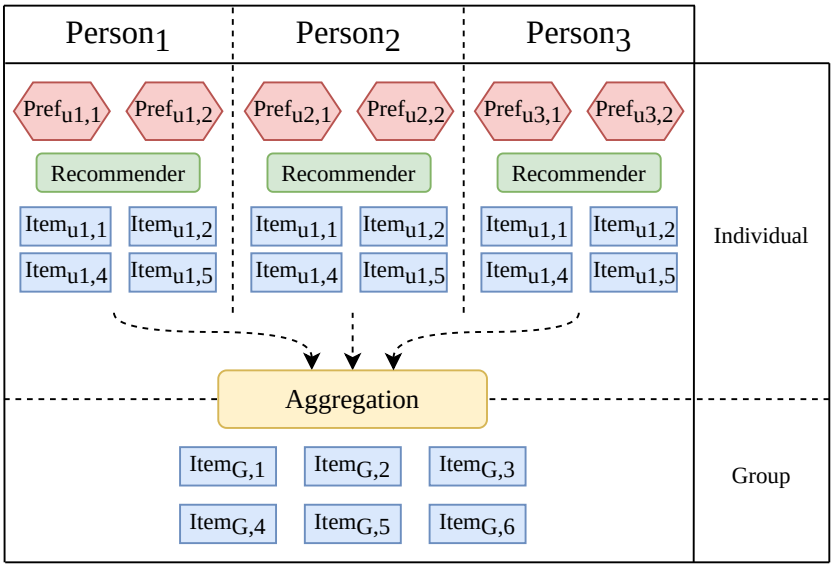


Figure 4.2: High-level overview of group recommendation with aggregation on top of recommendation results for individual users.

the consensus-based considers the preferences of all the group members, and the borderline-based considers a subset of the items by some limiting criteria.

We now list the most common aggregation methods as mentioned in [2], [14] and [43] specify to which of the group mentioned above they belong.

- **Additive utilitarian** (ADD, consensus)
  Sum of scores for an item across the group

$$\underset{i \in I}{\operatorname{argmax}} \sum_{u \in G} \operatorname{score}(u, i) \tag{4.1}$$

- **Approval Voting** (APP, majority)
  Number of users that like the item above a certain threshold

$$\underset{i \in I}{\operatorname{argmax}} \left| \{u \in G : \operatorname{score}(u, i) \geq \operatorname{treshold}\} \right| \tag{4.2}$$

- **Average** (AVG, consensus)
  Average of scores for an item across the group

$$\underset{i \in I}{\operatorname{argmax}} \frac{\sum_{u \in G} \operatorname{score}(u, i)}{|G|} \tag{4.3}$$

- **Average without Misery** (AVM, consensus)
  Average of scores for an item across the group only if the item is above a certain threshold for all group members

$$\underset{i \in I : \nexists u \in G | \operatorname{score}(u,i) \leq \operatorname{treshold}}{\operatorname{argmax}} \frac{\sum_{u \in G} \operatorname{score}(u, i)}{|G|} \tag{4.4}$$

- **Borda count** (BRC, majority)
  Sum of scores derived from item rankings. The ranking score is defined for each user by ordering the user's items by score and awarding points corresponding to the item's location in this ordered list. The worst item receives 1 point, and the best item $|I|$ points.

$$\underset{i \in I}{\operatorname{argmax}} \left(\sum_{u \in G} \operatorname{RankingScore}(u, i) \right) \tag{4.5}$$

  Where *ranking score* is defined as follows:

$$\operatorname{RankingScore}(u, i) := \left| \{ i_{other} \in I : \operatorname{score}(u, i_{other}) \leq \operatorname{score}(u, i) \} \right|$$

- **Copeland rule** (COP, majority)
  Difference between the number of wins and losses for pair-wise comparison of all items

$$\underset{i \in I}{\operatorname{argmax}} \Big(\mathrm{W}(t, I - t) - \mathrm{L}(t, I - t) \Big) \tag{4.6}$$

- **Fairness** (FAI, consensus)
  Users, in turn, one after another, select their top item.

$$\underset{i \in I}{\operatorname{argmax}} \operatorname{score}(u_{current}, i) \tag{4.7}$$

  Where $u_{current}$ is the user selected from G for each iteration according to some (in most cases circular or ping pong) rule.

- **Least misery** (LMS, borderline)
  Uses the lowest received rating among the group members as the item's aggregated rating.

$$\underset{i \in I}{\operatorname{argmax}} \left(\min_{u \in G} \left(\operatorname{score}(u, i) \right) \right) \tag{4.8}$$

- **Most Pleasure** (MPL, borderline)
  It uses the highest received rating among the group members as the item rating.

$$\underset{i \in I}{\operatorname{argmax}} \left(\max_{u \in G} \left(\operatorname{score}(u, i) \right) \right) \tag{4.9}$$

- **Majority Voting** (MAJ, majority)
  Uses the rating given by the majority of the group's members. (Can only work on discrete ratings)

$$\underset{i \in I}{\operatorname{argmax}} \left(\operatorname{mode}_{u \in G} \left(\operatorname{score}(u, i) \right) \right) \tag{4.10}$$

- **Most Respected Person** (MRP, borderline)
  Uses rating proposed by the most respected member of the group.

$$\underset{i \in I}{\operatorname{argmax}} \operatorname{score}(u_{most\_respected}, i) \tag{4.11}$$

- **Multiplicative** (MUL, consensus)
  Multiplies all received ratings together.

$$\underset{i \in I}{\operatorname{argmax}} \left(\prod_{u \in G} \operatorname{score}(u, i) \right) \tag{4.12}$$

- **Plurality Voting** (PLU, majority)
  Each user has a set number of votes that get distributed. The item with the most received votes is selected.

$$\underset{i \in I}{\operatorname{argmax}} \left(\sum_{u \in G} \operatorname{VotesAwarded}(u, i) \right) \tag{4.13}$$

  Where *votes awarded* is some function that decides for each user how the available votes will be distributed among the items.

Some of the methods have an additional distinguishing factor: they are iterative calculations instead of a single calculation that returns the final ordering. Most notably *Fairness*, it calculates the next item purely from a currently selected user, users changing in iterations one after another. Another one is *Plurality Voting*, this technique can also be iterative, but it is not mandatory. After the top item is selected, we reset the calculation, therefore iteratively selecting top items. All the other ones are single-iteration methods, where the final list requires only one calculation.

Now we will show how these strategies work together with an RS to provide a complete group recommender system, which will serve as an introduction to the inner workings of such a system in order for us to continue towards more advanced aggregation methods.

4.2.2 Usage with recommender systems

We now need to distinguish between *model aggregation* and *prediction aggregation* due to, in most cases, very different inputs and required outputs from the aggregation.

Prediction aggregation

In most cases, we use them as *prediction aggregation* after the recommendation is relatively straightforward. In all (to us known) cases, RS returns recommended items together with some sort of rating of the recommendation. There are some applications where this presumption does not hold, such as building a group recommendation aggregation on top of a *black box* recommender that only returns a list of items, for example, if extracting recommended items data from a web page where we only retrieve the list itself and we do not have access to the underlying recommender system. Nevertheless, in the following section, we will assume that we have full access to the rating data of the outputted items. We further assume that, if necessary, we can acquire ratings and an ordering for all possible items.

As shown in Figure 4.2, we first get a list of recommended items (together with ratings) for each user. Then we can directly apply the methods mentioned in Subsection 4.2.1.

Model aggregation

Running aggregation on top of user preferences, instead of lists of recommended items, is more challenging due to the many possible inputs the recommender can take, such as explicit or implicit feedback. As discussed in Section 4.1, we have two main options, aggregation of ratings and aggregation of mined preferences.

The first option is the aggregation of ratings, in other words creating a set of ratings representing the whole group. This approach is preferred due to its simplicity. We can use many of the previously mentioned simple methods, mainly the *consensus-based*, which aggregate ratings directly (positive and negative). Other mentioned methods can be used as well. However, some of them do not make much sense to adapt, for example, the voting-based, with which we will be very limited due to the usual sparsity of the feedback ratings that we have available.

The second option, the aggregation of mined preferences, is more present in content-based systems. We aggregate not the rating but some other data that represent the users' likes and dislikes. As an example, we will use the *tf-idf* keyword extraction which is by far the most popular method used in the text-based recommendation as mentioned in [44]. We can weight *tf-idf* concepts extracted from the user feedback by the received rating and then aggregate these concepts together for the whole group. As we can see, it is a more elaborate approach that requires more modifications to the simple methods mentioned.

4.3 Advanced methods

As we can see, the simple methods mentioned before are pretty straightforward. They have mostly straightforward and clear objectives and try to optimize in different ways, some quite vaguely specified goals. However, as discussed in Chapter 3, the objective is quite hard to define and measure. We will now introduce other methods that either directly or indirectly try to optimize some better-specified goal.

4.3.1 GFAR

So far, we have introduced methods that do not directly specify any form of optimization in the direction of *fairness*. A new method introduced in[45] directly optimizes the resulting list to balance the relevance of the recommended items in a rank-sensitive way. Similarly to all before mentioned methods, it is an aggregation approach that works on top of a single user RS.

Introduction

As mentioned, the authors focus on the *fairness* of $topN_G$ (*fairness* of the top N recommended items for group G). Where they define *fairness* as a property of the top-N_G items, not just any single item but the whole list. As already discussed in Section 2.1, the difference between optimization independently item after item versus in some way for the whole list can yield significantly better results when used in sequential consumption of items or balancing users with different preferences.

In the latter case of a member with differing preferences, we have more freedom to compensate and give more priority for balancing fairness towards this one user if we define fairness as a property of the whole list.

This kind of fairness is the main focus of the GFAR algorithm. In a way, it exceeds this by trying to optimize each prefix of the recommendation list. We will later define categories based on this fairness perception in Chapter 6. The GFAR algorithm is to us, the only known algorithm that optimizes this kind of prefix fairness we call *rank-sensitive* fairness.

In Subsection 4.2.1, we have already mentioned the FAI algorithm that tries to balance the fairness of the whole list. It generates the aggregated results in steps, each following step maximizing the utility of the following user in turn. This approach solves the problem mentioned above very simply by not considering the group preferences at all. It is just trying to please everyone in turns, not considering if the rest of the group will like the item in that turn.

Approach

What if we have instead considered the previously recommended items when selecting the next one in the list? The authors propose defining *fairness* together with ordering within the group by saying that "a *topN* is fair to a group if the relevance of the items is balanced across the group members for *each prefix* of the $topN_G$. In other words, for each iteration, we try to select an item that improves the balance as much as possible.

Let us first define Borda relevance following [15] as:

$$\text{Borda-rel}(u,i) = \left|\left\{j : \text{rank}(j, \text{topN}_u) < \text{rank}(i, \text{topN}_u)\right\}, \forall_j \in \text{topN}_u)\right| \quad (4.14)$$

Where $topN_u$ is a list of top N items for user u, rank is the position of item i in the user's u candidate list (position is determined by ordering the items based on the returned items score, from the best to the worst, where the best receives a rank of 1). Borda relevance essentially gives zero points to the last item and increases the points by one for each position up in the list, with the first/top item getting $(N-1)$ points. We can calculate Borda-rel from rank simply by $\text{Borda-rel}(u,i) = N - rank(u,i)$. Note that if there are items that have the same rank, then the maximum value has to be awarded to the group in order to translate this way to Borda-rel.

Now we can set the probability of item i being relevant for user u as:

$$p(\text{rel}|u,i) = \frac{\text{Borda-rel}(u,i)}{\sum_{j \in \text{topN}_u} \text{Borda-rel}(u,j)} \quad (4.15)$$

Let also $p(\neg\text{rel}|u,S)$ be the probability that item i is not relevant for any of the items in set S. We derive the probability that at least one item from set S is relevant to the user u as:

$$\begin{aligned} p(\text{rel}|u,S) &= 1 - p(\neg\text{rel}|u,S) \\ &= 1 - \prod_{i \in S} (1 - p(\text{rel}|u,i)) \end{aligned} \quad (4.16)$$

Now we want to generalize for the whole group, so we define $f(S)$ as the sum of probabilities for each user that they find at least one relevant item in the set S:

$$\begin{aligned} f(S) &= \sum_{u \in G} (1 - p(\neg \text{rel} | u, S)) \\ &= \sum_{u \in G} \left(1 - \prod_{i \in S} (1 - p(\text{rel} | u, i)) \right) \end{aligned} \tag{4.17}$$

We have used Group G as a constant in Equation 4.17, and forward, the group will stay the same during the calculation. $f(S)$ shows how to balance the fairness amongst the group members for a specified set of items. We now want to extend it in order to make it rank-sensitive by defining a marginal gain in function f for adding a new item i to the set S as follows:

$$\begin{aligned} f(i, S) &= f(S \cup i) - f(S) \\ &= \sum_{u \in G} \left[p(\text{rel} | u, i) \prod_{j \in S} (1 - p(\text{rel} | u, j)) \right] \end{aligned} \tag{4.18}$$

Where we have used Equations 4.16 and 4.17, finally, we can define an ordered set that is considered fair if it balances each of its prefixes. In other words, the first item of the set should be considered fair/balanced by all group members as much as possible, then the first two items, and so on. We define fairness in an ordered set OS as:

$$\text{fair}(OS) = \sum_{k=1}^{|OS|} f(OS)[k], \{i \in OS : \text{rank}(i, OS) < k\}) \tag{4.19}$$

4.3.2 XPO

The last algorithm performed a greedy search based on the probability of at least one item being relevant for each user in order to find balance among users in a rank-sensitive way.

The following method introduced in [46] XPO and its variant NPO are aggregation approaches that explore a simple but intuitive notion of fairness that utilizes Pareto optimality of item's raking in all admissible ways in which a group may reach a decision.

Introduction

The authors specifically aim to minimize the feeling of dissatisfaction, which somehow differs from the majority of previous works that aim to optimize the group's overall satisfaction or, more recently, from works that aim to optimize fairness for each group member.

With a measure utility for each pair of users and item, we have a group utility to be the average member utility according to [47] and fairness to be the minimum member utility, which is equal to the Least misery approach.

With these definitions, the measure of utility itself becomes important as it is the main factor that we are optimizing on. If we have a recommender system

that recommends top-K items for each user, then for a set of recommended items N, the utility of that user is the similarity between the top-K list and list N.

There are multiple similarity measures for comparing two lists, such as symmetrical measures, Spearman's foot rule, rho, and Kendall's tau. Furthermore, unsymmetrical, better-known precision, recall, and normalized discounted cumulative gain.

The authors' approach is based on the notion of Pareto optimality. Pareto optimal item for the group is an item that ranks the highest according to all group members. The group unanimously agrees that no other item would be sharply better than this one.

Therefore this Pareto optimality is inherently fair due to all members considering the item the best. However, most usually, items are always a different set of trade-offs that need to be balanced.

Let us now be more specific and present the algorithms' details.

Approach

The authors assume that the system is providing us with a top-N list of recommendations for each member m of a group $\mathcal{G}$. This set of items is considered *ground truth*. Each item is considered a vector in m-dimensional space $\mathbb{R}^n$, where each dimension is a rank $r_u(i)$ of the item in the user's top-N list of items. If an item is not in that list, then rank of N+1 is awarded. For completeness, rank is a position in the ordered list of top-N items of each user, therefore the best item has a rank 1, second best has a rank of 2 and the last item has a rank N. We assume this rank for each user independently only based on theirs top-N items.

For a group we get candidate items as

$$\mathcal{C}_\mathcal{G} = \bigcup_{u \in \mathcal{G}} topN(u). \tag{4.20}$$

Further, we say that an item i dominates item i' according to a group G if

$$\forall u \in \mathcal{G} : r_u(i) \leq r_u(i') \wedge \exists u' \in \mathcal{G} : r_{u'}(i) < r_{u'}(i') \tag{4.21}$$

It means that ranks for all users must be equal or better, but for at least one user, the rank needs to be sharply better.

A good question might arise, why the equality even matters if items are ranked based on their position in a sorted list? Two situations can arise. Either the utility is exactly the same, and therefore the items will be in the same position, or the items are not in the top-N for that user. In the latter case, we award the rank of N+1 as already mentioned, and there can, and most probably will be, many items sharing this lowest rank.

Further, when evaluating an aggregation strategy, we say that it is Pareto-efficient if whenever an item i is ranked higher than another item i' by each group member, then also the strategy rates the item i higher than item i'. Pareto optimal strategy respects the Pareto domination of items described before.

This property Pareto-efficiency is respected by all rating and rank aggregation strategies. We can easily see that if a strategy recommends an item that is Pareto dominated by another one (is better, by rank and or rating), then it has recommended a subpar item and therefore is not optimal, based on the underlying rating.

We call a set of items that are not dominated by any other items Pareto optimal (PO). One interesting side note that the authors do not address: does always at least one Pareto optimal item exist? We see that PO property is transitive, therefore if an item i dominates a set of some items I, then if there is an item i' that dominates i, then from the transitivity, it also dominates all items from the set I. And so on, therefore there always exists an item or a set of items that are Pareto optimal. In a different way, we can view this relationship as an acyclic-directed graph.

Next, they define *N-level Pareto optimal* (N-lPO) set, which contains items that are dominated by at most $N-1$ other items. Note that N-lPO set contains items from all smaller PO sets, such as *3-lPO* $\subset$ *2-lPO* $\subset$ *1-lPO*.

We can calculate the N-lPO level for each item by comparing it with all other items from the set and directly calculating the number of times other items dominated the current one. This, unfortunately, leads to $\mathcal{O}(n^2)$ time complexity, where n is the size of the candidate set. It would be interesting to investigate the possibility of using the transitivity property cleverly to get better time complexity, at least in the average case, but that is outside of the scope of this work.

Ideally, we would want to take the N-lPO from all the items, but that would require to first have a ranking of all items for each user, and then calculating the Pareto optimally level between all pairs of items, which would be very computationally expensive. We, therefore, take an approximation of the N-lPO set of items from the candidate set where for each user we request their top-N where N is larger than the final required recommendation size or as large as computationally feasible.

The next step is the interesting part of the authors' new method, they consider all *linear aggregation strategies*, where linear strategy assigns weight w_u to each user so that the total weight sums to 1, then item raking is multiplied user-wise with this weight and summed up as

$$1 = \sum_{u \in \mathcal{G}} w_u. \tag{4.22}$$

Then items score under this linear strategy $\mathcal{L}_w$ is

$$S_{\mathcal{L}}(i) = \sum_{u \in \mathcal{G}} w_u r_u(i). \tag{4.23}$$

Each linear aggregation strategy can be uniquely represented by the weight w.

The motivation behind considering all linear aggregation strategies is to get the ratio for items that do not dominate each other. When two items lie on the same Pareto level, it does not mean they achieve the same average score over all l. strategies. We can use this ratio to further distinguish between the items and therefore compare items even when they are on the same Pareto level. We can see an illustration for 6 items and two users in Figure 4.3. Item i_1 and i_2 lie on the Pareto front and therefore there is not clear which item is better. But for only one linear strategy in this case, with weight w the items will have the same score. For any other weight vector, there will be a clear winner. We, therefore, want to compute this ratio. We can compute this two-dimensional case analytically as $3/7$ and $4/7$ for weight $w = (3/7, 4/7)$. We can see that only Pareto optimal

items would have a non-zero probability, and for every non-Pareto optimal set of items, there would be a clear winner.

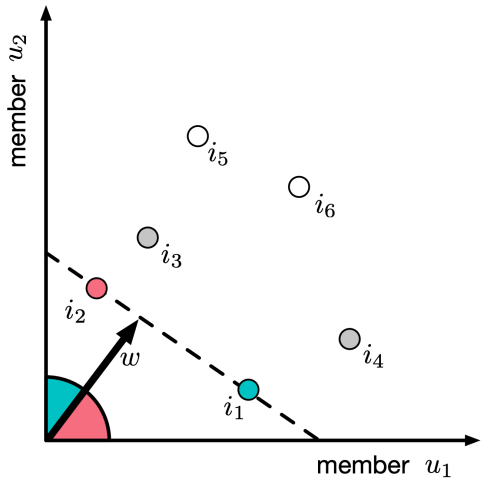


Figure 4.3: Example of 6 items based on two group members u_1 and u_2 from [46]. Item i_2 is Pareto optimal for user u_1, and item i_1 is Pareto optimal for u_2.

In 2 dimensional case, computing the exact probability is possible, but in a multidimensional case, the problem does not have a simple analytical solution. Therefore considering all possible weights is unpractical in order to get an average of items rating across all linear strategies, we opt for a Monte Carlo method of computing this probability.

Therefore, we generate many random weights, each representing a different linear aggregation strategy. For each $\mathcal{L}_w$ we compute how many times an item i ended up being in the *top-N* where N, in this case, is the desired size of the recommended list.

Items are then rated by how many times each N-PO item ranks within the *top-N*. Best N items are returned as the output of the group recommender strategy. We call this approach N-level Pareto optimal aggregation (NPO).

The size of N-level Pareto optimal items can be much larger than the required size of output N, we therefore can perform a binary search to identify the smallest $x \in [1, N]$ for which there are at least the desired amount of items N. We can formulate this as

$$argmin_{x=1}^{N} |\mathcal{P}_x| \geq N, \tag{4.24}$$

where $\mathcal{P}_x$ represents the set of *x-level Pareto optimal* items. We call this aggregation strategy XPO.

Note, the only difference between NPO and XPO is the smaller candidate final candidate selection step. The initial candidate list drawn from top-N candidates for each user $\mathcal{C}_\mathcal{G}$ stays identical.

4.3.3 D'Hondt direct optimization (FuzzDA)

This method directly uses a definition of fairness that is widely around the world as a mandate allocation strategy for elections and the division of a discrete number of mandates/items.

It focuses on D'Hondt's algorithm (DA) which is commonly used in European elections. DA is a greedy selection algorithm that minimizes the number of votes that need to be left aside so that the remaining votes are represented exactly proportionally as described in [48].

The procedure behind DA works as follows: in steps, it awards the party with the largest quotient *quot* one seat, and after each step, the quotient is recalculated. The quotient is calculated as follows:

$$quot(V_p, s_p) = \frac{V_p}{s_p + 1}, \tag{4.25}$$

where V_p is the total number of votes that party p received and s_v is the number of seats that have been allocated to party p so far (initially 0).

In order to compensate for this, FuzzDA presented in [49], can be used. For each group member u and each candidate item i, we assign a relevance score of that item for that user to $r_{u,i} \in [0, 1]$. At each step, FuzzDA select a candidate that maximizes $\sum_{u \in G} quot(V_u, s_u) * r_{u,i}$. After each step, each group members' accountable votes are decreased by the factor of $r_{u,i}$. This represents a fuzzy approach that fixes the DA problem of being unable to handle items that are relevant to multiple parties by lowering their accountable votes (therefore the power in the next step) by how much utility they have received with each new item.

5. Datasets

People are gregarious in nature, but the same, unfortunately, cannot be said about machine learning datasets. The vast majority of them are not directly usable in the group RS research due to only containing information about single-user preferences. To design and evaluate group recommender systems, we preferably need datasets that contain information about groups' preferences.

In this chapter, we will describe which datasets are suitable for use in the RS domain. We describe commonly used datasets in the non-group RS domain. Then, we analyze their high-level properties and describe what transformations are needed to make the dataset convenient to use.

Next, we will talk about the existing group RS datasets and introduce methods that can be used to generate the group recommendation information synthetically from non-group RS datasets. We will use these methods to generate standardized synthetically enriched group RS datasets from non-group datasets that we describe in the single-user datasets subchapter.

5.1 Single user datasets

Multiple well-known and thoroughly studied datasets exist in the recommender system domain. Let us present the popular ones that seem to be utilized the most.

When we talk about the specific format of the data, then we are referring to the unified format which we have transformed the original data into using the dataset transformation library. Further description of the data format transformations follows in Subsection 5.1.6.

5.1.1 Movie Lens

One of the most well-known datasets in the RS domain, it contains 25 million ratings in total across 62,000 movies and 162,000 users. The data were collected between 1995 and 2019, and the current version of this size (25M) was released in November of 2019. Data are organic and come from a web-based recommendation system at movielens.org. The project was specifically created in order to gather research data on personalized recommendations by researchers at the University of Minnesota.

The dataset is in a suitable format that is easy to parse and use. A further description follows in Subsection 5.1.6.

Number of items: 62,000
Number of users: 162,000
Number of user-item interactions: 25,000,095
User-item interactions format: Sparse matrix of ordinal ratings [1, 1.5, 2, ... 4.5, 5] - user rated a movie
List of data tables: Movies (detail in Table 5.1), Ratings (detail in Table 5.2), Tags, Links, Genres, Genome Scores, Genome Tags

item_id	title
1	Toy Story (1995)
2	Jumanji (1995)
3	Grumpier Old Men (1995)
...	...
209169	A Girl Thing (2001)
209171	Women of Devil's Island (1962)

[62423 rows x 2 columns]

Table 5.1: Short snippet of Movie Lens dataset's `movies.csv` table.

user_id	item_id	rating	timestamp
1	296	5.0	1147880044
1	306	3.5	1147868817
1	307	5.0	1147868828
...	...	...	...
162541	58559	4.0	1240953434
162541	63876	5.0	1240952515

[25000095 rows x 4 columns]

Table 5.2: Short snippet of Movie Lens dataset's `ratings.csv` table.

5.1.2 KGRec

KGRec is a smaller and less known dataset. We have chosen this dataset because it was utilized in the GFAR method introduced in [45] and described in Subsection 4.3.1. This dataset consists of two separate datasets of music and sound, KGRec-music and KGRec-sound, respectively.

The first music dataset comes from songfacts.com (items and text descriptions) and last.fm (ratings, items, tags). Each user-item interaction is a user listening to a song.

The second sound dataset comes from freesound.org. Items are sounds with descriptions using text and tags created by the person who uploaded the sound. Each user-item interaction is a user downloading an item, in this case, a sound.

Further, we will consider only the music dataset and not utilize the sound dataset. We have made this decision to simplify comparisons due to the origin of the sound dataset itself. It comes from a web page where users can upload and download random sounds of their choosing, such as the 'Mechanical clock movement' sound, 'Industrial elevator' sound, and other. The need for these sounds is most probably driven by people using them for their profession, such as video production, and therefore does not reflect natural content consumption preferences.

Both datasets were created for the needs of [50], where they were introduced, and they are altered for the needs of research in Recommendation Knowledge Graphs. Further, the original data that was used for the creation of these datasets are described in [51].

Number of items: 8,640; 21,552[1]
Number of users: 5,199; 20,000
Number of user-item interactions: 751,531; 2,117,698
User-item interactions format: one-valued implicit feedback - user listened or downloaded a song/sound
List of data tables: Ratings(detail in Table 5.3), Tags, Descriptions

user_id	item_id
7596	68
7596	130
7596	330
...	...
50572897	8618
50572897	8619

[751531 rows x 2 columns]

Table 5.3: Short snippet of KGRec dataset's `music_ratings.csv` table.

5.1.3 Netflix Prize

Data that were originally released in 2009 by the Netflix.com video streaming company for the *Netflix Prize*, an open competition with the main prize of 1 million dollars. It contains data from more than 400 thousand randomly selected users from the company's database. Data contain information about users' ratings of movies. It was originally available on the contest web page but has been removed.

The original data was split into multiple files in a file for ratings per movie manner. Each rating is a quadruplet of the form '<user, movie, date of the rating, rating>'.

Number of items: 17,770
Number of users: 480,189
Number of user-item interactions: 100,480,507
User-item interactions format: sparse matrix of ordinal ratings [1, 2, 3, 4, 5]
List of data tables: Ratings (detail in Table 5.4), Movies (detail in Table 5.5)

user_id	item_id	rating	date
6	30	3	2004-09-15
6	157	3	2004-09-15
6	173	4	2004-09-15
...	...	...	...
2649429	17627	3	2003-07-21
2649429	17692	2	2002-12-07

[100480507 rows x 4 columns]

Table 5.4: Short snippet of Netflix dataset's `ratings.csv` table.

[1]All KGRec statistics are in order - music dataset; sound dataset

item_id	release_year	title
1	2003.0	Dinosaur Planet
2	2004.0	Isle of Man TT 2004 Review
3	1997.0	Character
...	...	...
17769	2003.0	The Company
17770	2003.0	Alien Hunter

[17770 rows x 3 columns]

Table 5.5: Short snippet of Netflix dataset's `movies.csv` table.

5.1.4 Spotify - Million Playlist Dataset

This dataset was released in January 2018 for *The Spotify Milion Playlist Dataset Challenge*. It contains 1,000,000 playlists with information about tracks that are part of each playlist. The primary purpose of this dataset was to study and develop better algorithms for automatic playlist continuation where the system could recommend songs that are similar to those already in the playlist. In contrast to the Netflix challenge, no prize was to be awarded at the end of the challenge.

Even though the context of this dataset is playlists and not users, we utilize a different view of the dataset, where each playlist will represent a single user. This way, we have another big and organic dataset at our disposal. It can therefore be used not only for playlist continuation tasks but also for the classical RS domain tasks. In a sense, a single playlist is a specific subset of the user's preference that has created the playlist. Therefore we expect to see a narrower preference distribution for each of these 'playlist' users.

For completeness, it is necessary to add that some playlists are 'collaborative', meaning that they were created by multiple users. Nevertheless, they account for only 2.3% of all playlists, which in our opinion, does not substantially affect the dataset. These collaborative datasets could be used as a group recommender dataset on their own. Unfortunately, the information about which user added which track to the collaborative playlist is not present.

playlist_id	item_id
549000	0
549000	1
549000	2
...	...
302999	133087
302999	133088

[66346428 rows x 2 columns]

Table 5.6: Short snippet of Spotify Milion Playlist dataset's `ratings.csv` table.

Number of items: 2,262,292
Number of users: 1,000,000
Number of user-item interactions: 66,346,428

item_id	item_name	artist_name
0	Boots of Spanish Leather	Bob Dylan
1	Mr. Tambourine Man	Bob Dylan
2	Danny's Song	Loggins & Messina
...	...	...
2262290	Robin Hood	Crazy Fool
2262291	Guilttrip	Ace Reporter

[2262292 rows x 6 columns]

Table 5.7: Short snippet of Netflix dataset's `tracks.csv` table. (Columns `item_uri`, `artist_uri`, `album_uri`, containing URI to Spotify object were omitted for simplicity due to their substantial length.)

User-item interactions format: one-valued implicit feedback - user added a song to a playlist
List of data tables: Tracks (detail in Table 5.7), Ratings (detail in Table 5.6)

5.1.5 Comparison of datasets

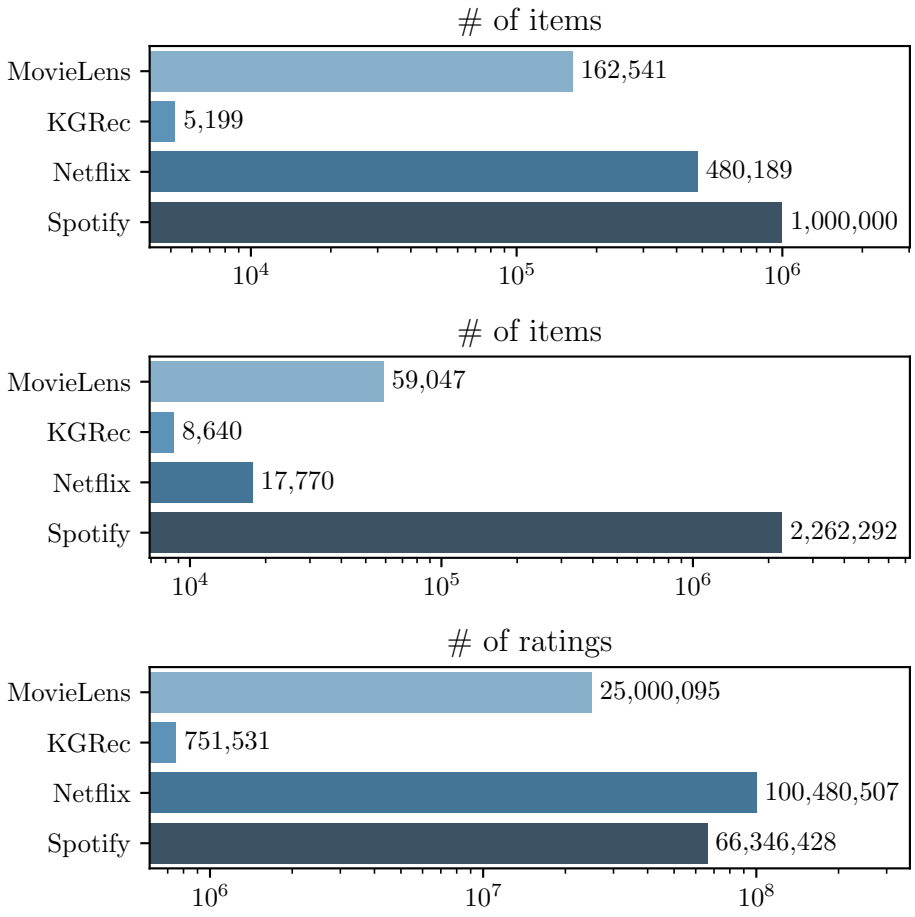


Figure 5.1: Size comparison of the selected datasets. All x-axes are log scales due to the big differences between the dataset.

We have described each dataset separately. Let us now compare them together to see how they differ. As shown in Figure 5.1, the Spotify and Netflix datasets are the biggest. We see that the KGRec dataset is almost two orders of magnitude smaller than the rest. As already mentioned, we have selected it due to its common utilization in the related literature. Spotify dataset is different by having two orders of magnitude more items, which can present a challenge on its own. For example, if we use matrix factorization methods to compute the preference, then the amount of memory will rise by two orders of magnitude as well. Additionally, the sparsity of ratings is higher, which can negatively affect efficacy.

The Movie Lens dataset has a potential benefit as it has actual ratings for each user-item interaction. All other presented datasets only contain user-item interactions in the form of one-valued implicit feedback.

In Figure 5.2, we present the distribution of ratings among users. The left y-axis, blue, presents the count of users with a particular number of ratings. We have clipped the users with a high number of ratings so that we can better see the interesting part of the data, which the long tails would otherwise squash. The clip was made on the last 10% mass of all ratings.

We see that Movie Lens nicely follows a power-law distribution. In this dataset, only users with more than 20 ratings are present, which is also visible in the figure and is most probably why we do not see the same initial rise in ratings as for Netflix and Spotify datasets'.

The KGRec dataset is different. It is more jagged due to the smaller effect of being smoothed out by the amount of data as with the other datasets. Interestingly, the number of users does not follow an exponential distribution in contrast to the other datasets. At first, we thought the reason was that the dataset was gathered at a website where users listen and download songs. These songs are always a part of an album; if a user is downloading one song from an album, they will most likely also download the rest of the album. However, that would not explain why this number of ratings per user starts at around 75.

The actual reason for this can be partially found in the original paper - [50]. As already described in Subsection 5.1.2, the dataset was altered to better fit the required research objective of recommendation using knowledge graphs. As such, songs with less than ten interactions have been removed, users with less than 50 item interactions have been removed, and only songs with over-average plays were counted as user-item interactions. Nevertheless, this would not explain the nonexponential nature of the distribution. We have downloaded and explored the original dataset from [51], the original dataset is not only one-valued implicit feedback, but it is the number of times a user has played the song. When we visualize the original dataset using the 'sum of plays per user' instead of the 'count of interactions per user', we get a natural-looking exponential distribution. Therefore, the most probable reason for the KGRec's dataset distribution is that users like to replay a smaller number of songs multiple times, creating more of a normal-looking distribution.

Further, the Netflix dataset looks similar to Movie Lens, with two exceptions. The first is that Movie Lens includes only users with at least 20 ratings. Therefore the initial increase in the number of ratings is not visible. Secondly, both Netflix and Spotify datasets have cumulative rating distribution shifted more to the right, which means that there are more ratings among users that were more active on

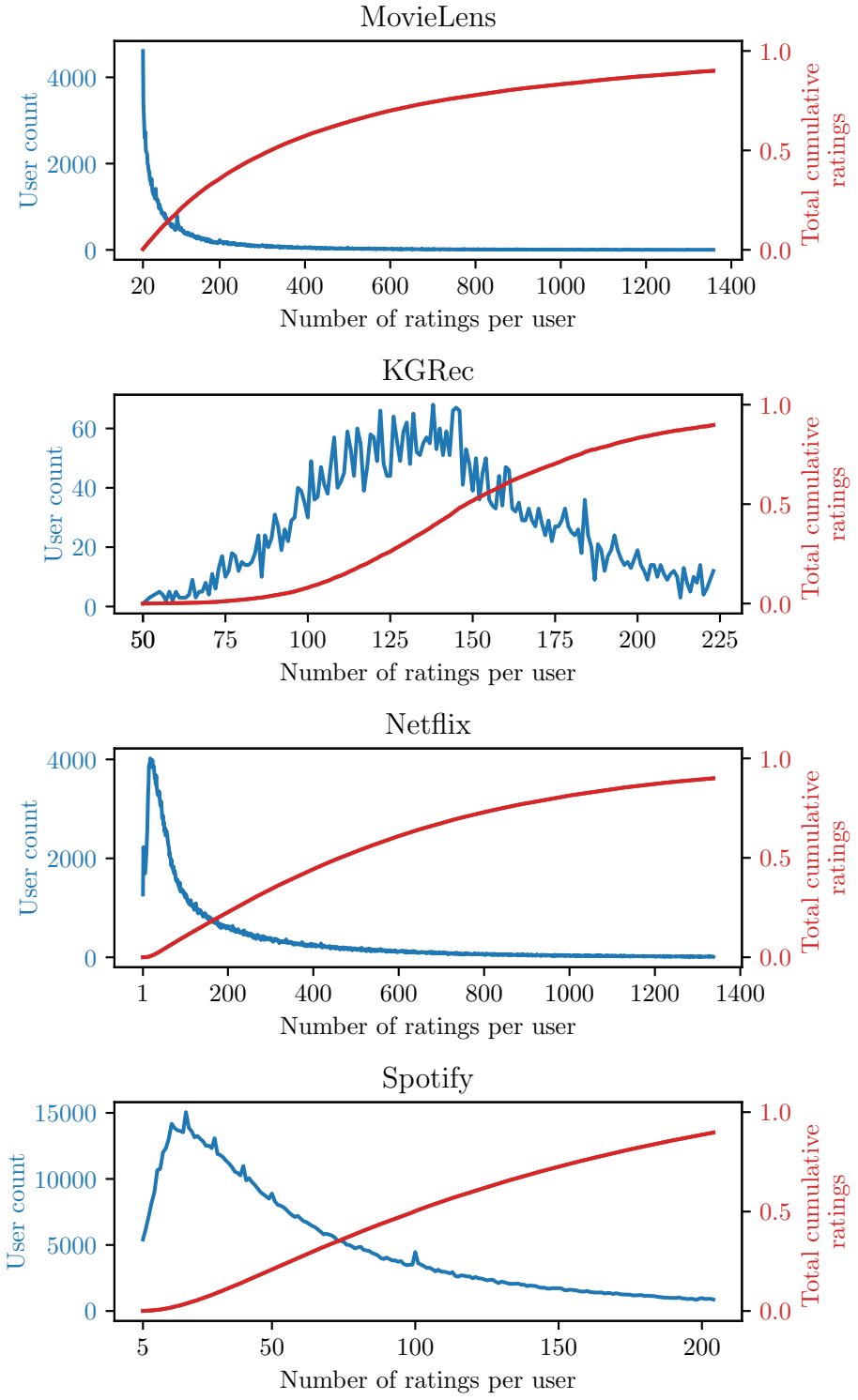


Figure 5.2: Distribution of ratings for groups of users by how many ratings they have made. The blue line shows how many users have made that particular amount of ratings, while the red line shows the total cumulative mass distribution of ratings. We have clipped the number of ratings by total cumulative ratings of 90% due to very long-tailed data, where some small amount of users created a very high amount of ratings.

the service and interacted with many items.

5.1.6 Datasets gathering and processing

Processing the datasets mentioned above was more difficult than it should have been. They are not easily accessible. Some are available only behind a login wall and in different, incompatible, and non-standard formats. We have therefore processed and unified these datasets using a tool that we have developed. At first, we wanted to create shared storage where they would be available in the already transformed form, but that is not feasible due to the datasets' licenses. Let us now describe the data transformations we have done for each dataset. We aim to have the datasets in standard zipped CSV format that can be simply loaded by most of the widespread data manipulation tools such as Pandas. Additionally, if not misleading, we prefer the columns of the datasets to be named in a unified fashion.

- **Movie Lens** dataset is available at the the authors web page grouplens.org/datasets/movielens/25m/. This dataset is easily accessible and ready-to-be-used (files in valid CSV format, zipped together in a single archive). This was the nicest dataset to start working with. No transformations were necessary apart from remapping user and item ids to be sequential.

- **KGRec** dataset is available for download at the authors web page upf.edu/web/mtg/kgrec. Unfortunately, downloading the dataset is not straightforward because the download link is an unsecured HTTP on a secured HTTPS site. This is a problem while using modern browsers, which do not support mixed HTTP and HTTPS content. The dataset has ratings in a standard CSV form with redundant information about the incidence, which is always of value 1. Main data in sparse incidence matrix representation are in the form of '<userId, itemId>'. Additional data with tags and descriptions of items are separated into individual files in the original dataset. We have transformed them into two CSV tables of form '<itemId, tags>' and '<movieId, description>' respectively. Further, remapping of user and item ids to sequential values was performed.

- **Netflix** dataset is available on an independent web page kaggle.com/datasets/netflix-inc/netflix-prize-data. The original web page of the challenge is no longer available. The uploader additionally processed the dataset by aggregating the small per-movie rating files into four bigger files. This dataset was in a non-standard format where ratings were not in CSV but in a custom format reflecting the original movie ratings per file division. Each group of ratings for a movie starts with a line only containing the id of the movie and a colon, then ratings for the movie follow each per line in a format '<user-id,rating,timestamp>'.

- **Spotify** dataset is available at AIcrowd.com, where the original Spotify challenge was introduced. Unfortunately, the dataset is behind a login wall. After registering and logging in, it can be downloaded from aicrowd.com/challenges/spotify-million-playlist-dataset-challenge.

The dataset is comprised of 1000 JSON files, each describing 1000 playlists. The dataset contains a lot of information about each playlist such as the name, when it was modified, how many followers it has, and so on. We go through all data and simultaneously create a list of playlists to track mapping and another list with any additional track information, such as the Spotify URI links for the track, album, and artist. We have created a separate numerical track id due to the native one of the track URI being too long. Apart from the JSON files, the zip archive contains some python code to provide easy parsing and simple statistics about the dataset.

We have created a Python CLI tool that can be used to download the available datasets easily. This tool is available in our Git repository at github.com/LadislavMalecek/MasterThesisAnalysis. Run `./gather_datasets/download_and_transform.py` to execute the script. All relevant information about the supported parameters is available when run with the standard '–help' switch.

5.2 Group datasets

Let us now investigate datasets that contain information about groups of users. We will look through some of the datasets mentioned in related literature.

5.2.1 Datasets overview

When reading through related literature on GRS, we have mostly encountered synthetically created datasets either from the Movie Lens dataset or the Netflix dataset. The main reason for not utilizing datasets with innate group information is that not many of them exist. Let us go through some of the datasets mentioned in the related literature and determine their suitability for group recommender system research.

First, we would like to point out a critical flaw that most literature in our domain has. Suppose the authors decide to use some small, unknown, and or proprietary dataset, instead of using the traditional Movie Lens and Netflix datasets. Then for the research reproducibility, it is necessary to provide a detailed description about where and how the dataset can be obtained and how the raw dataset was created, filtered, and altered. Most authors do not mention these details and only mention which dataset they were using and some high-level information about the dataset, such as the number of users, items, and interactions. This makes the papers' experiments irreproducible and unverifiable. The same can sometimes be said about papers that use single-user datasets with synthetically generated group information. Sometimes, it is not entirely clear how the synthetic groups have been generated and to what values the parameters have been set.

In Table 5.8, we present a high-level overview of datasets that we found in the related literature.

	#users	#items	#groups	avg. g. size
CAMRa2011[52]	602	7,710	290	?
Douban[53]	23,032	10,039	58,983	4.2
Gowalla[53]	25,405	38,306	44,565	2.8
Mafengwo[52]	5,275	1,513	995	?
Meetup[54]	42,747	2,705	13,390	16.66
Plancast[55]	41,065	13,514	25,447	12.01
Yelp[53]	7,037	7,671	10,214	6.8
Weeplaces[53]	8,643	25,081	22,733	2.9

Table 5.8: Size overview of selected GRS datasets. Note: Some dataset sizes are given after transforming the original datasets by removing users and items that are not part of any group.

CAMRa2011

The CAMRa2011 dataset was released in 2011 for the 2011 ACM Recommender Systems Conference. The dataset is unavailable at the original location, and we could not retrieve it from the web. Numbers in Table 5.8 are taken from [52]. We have found an altered version of the dataset at github.com/LianHaiMiao/Attentive-Group-Recommendation, which is a GitHub repository for the code and experiments from [52]. The dataset is quite small and we are doubtful about its quality and source.

Douban

This dataset has similar problems to the CAMRa2011 dataset. We were unable to retrieve it from the web. At the same time, in the already mentioned repository for [52], we found an author's comment about adding the dataset soon (in 2018), which was never done.

Gowalla

Gowalla was a location-based social network. It was dissolved in 2012 when Facebook acquired it. The dataset contains people logging in to locations that they have visited. The dataset can be easily downloaded at yongliu.org/datasets/. We have discovered this page and dataset in [53].

This dataset does not directly contain any group information. However, it could be inferred by combining the check-in data in the format '<userId, placeId, datetime>' and friendship data that link pairs of users '<userid1, userid2>'. If a person and some of their friends visit the same place around the same time, we can state that they were probably all part of the same group. Moreover, if they visit multiple places together, then the chance of a random occurrence drops significantly.

We suspect that there will be a substantial location similarity bias due to the data being location-based. If someone visits a specific place, they are very likely to visit a popular place nearby regardless of its quality.

- **Available group information:** No explicit information, but group information can be interpolated from information about friendships and user-

item interaction timestamps

- **User-item interactions:** one-valued implicit feedback
- **Group-item interactions:** none
- **Items type:** points-of-interests

Mafengwo

Tiny and proprietary dataset mentioned in [52]. The dataset is unavailable for download. We were unable to find another source from which we would be able to download this dataset.

Meetup

This dataset is crawled data from website meetup.com from [54] used in [55]. Meetup is a popular web page for meeting other people with similar hobbies and interests. This dataset contains only data from two regions, New York City and the state of California. One substantial distinction from other datasets is that the items represent non-repeating social events. This creates difficulties with similarity calculation between users due to the low average number of user-item interactions per item. Groups are defined explicitly with the grouping function, and members can communicate within the group and attend events or plan events together.

With some difficulties, we have downloaded the dataset from personal.ntu.edu.sg/gaocong/datacode.htm.

- **Available group information:** group memberships, groups are on average big
- **User-item interactions:** one-valued implicit feedback
- **Group-item interactions:** none
- **Items type:** social events

Plancast

Unfortunately, we were unable to obtain this dataset for further analysis. In [55], where this dataset is mentioned, there is no download link, only a reference to [56], where no additional information about the source is provided.

Yelp

This dataset contains reviews for businesses and places. In [53], they use a subset of the whole dataset only for the city of Los Angeles. The whole dataset can be downloaded at yelp.com/dataset and, in its unfiltered variant, is vastly bigger than other mentioned datasets with over 6,9 million ratings.

- **Available group information:** no explicit information, but group information can be interpolated from friendships and user-item interaction's timestamps

- **User-item interactions:** one-valued implicit feedback and text reviews
- **Group-item interactions:** none
- **Items type:** Points-of-interests

Weeplaces

Similarly to the Gowalla dataset, Weeplaces was a website that aimed to visualize users' location-based check-in activities. It has been integrated with multiple location-based social networking services such as Foursquare and Facebook. It contains information about check-ins and friendships, the same as the Gowalla dataset. It can be downloaded from yongliu.org/datasets/. We have discovered this page and dataset in [53]. Again, the same arguments for the group information and location-based bias hold for this dataset, the same as for the Gowalla dataset.

How groups are created is not described in the original paper. Other information which is missing is a complete description of how was the raw dataset transformed and filtered. A high-level description is present, but it is incomplete.

- **Available group information:** none explicit, can be interpolated from friendship information and user-item interaction timestamps
- **User-item interactions:** one-valued implicit feedback
- **Group-item interactions:** none
- **Items type:** Places check-ins

Dataset selection

Let us now select a subset of the mentioned datasets for further analysis and for inclusion in our download and transform tool.

We have rated all datasets on a scale of 'poor', 'ok', and 'good' in Table 5.9 for the following important criteria:

- **Ease of retrieval**
  We award a 'poor' rating if the dataset cannot be downloaded at all. Award an 'ok' rating if we can download the dataset with some difficulties from either the source in the mentioned paper or any related linked papers, as well as if we can download the dataset from an unrelated source. We award a 'good' rating if the dataset is easily downloadable using the original sources or any source original to the dataset, such as the original research challenge web page.
- **Available group information**
  We award a 'poor' rating if the group information is either not very fitting to our use case, the dataset does not contain any, or the group information is very scarce. We award 'ok' and 'good' in cases where the information is present, and the quality is good or great, respectively. The ideal situation is if the dataset contains full information about which members were part of the group-item interaction and when the group-item interaction is rated by each member.

- **Size**
  We award 'poor' if the dataset size is borderline unusable (the definition of what size is and is not usable can differ widely based on the utilized methods). We award good if the amount of information is on the order of information we can find in single-user datasets. We award 'ok' to sizes in between.

- **Source legitimacy**
  We award 'poor' if the dataset comes from either a not well-known service or from a service that is already canceled. We award 'ok' if the source is less well-known but legitimate and easily traceable, and finally, we award 'good' if the source is a well-known service used worldwide.

Additionally, some criteria that we could not assess (due to the dataset not being available for download) were marked 'n/a'.

With all relevant information about each dataset found in the current subchapter and in Table 5.9, we have concluded that, unfortunately, no dataset currently satisfies our criteria. Gowalla, Weeplaces, and Yelp datasets are borderline usable due to retrieving the group information from friendships and timestamps of reviews. Constructing the groups would require us to set a window parameter, either floating or fixed, to group users together. Further, the Meetup dataset can be used, but the average group size is unnaturally high, especially for researching fairness which in the context of big groups becomes less relevant and harder to satisfy.

	ease of retrieval	available group information	size	source legitimacy
CAMRa2011	poor	n/a	poor	poor
Douban	poor	n/a	ok	poor
Gowalla	good	poor	ok	ok
Mafengwo	poor	n/a	poor	poor
Meetup	ok	poor	ok	good
Plancast	poor	n/a	ok	poor
Yelp	good	poor	ok	good
Weeplaces	good	poor	ok	ok

Table 5.9: Ratings of selected GRS datasets.

5.3 Creation of artificial groups

As we can see, datasets with group information are a scarce resource. Ideally, we would like to have a dataset that contains the following data - user-item interactions, information about groups that the users belong to, and group-item interactions.

However, this information is hard to obtain in practice. Most of the datasets that we have seen only contain information about friendships, not directly about groups. Further, they do not contain group-item interactions, which we would

like as the ground truth when training group recommender systems. Instead, they only contain user-item interactions from which we can only estimate the corresponding group-item interactions.

We cannot enrich the datasets with group-item interactions because that is the task we aim to solve. We would, in a way, just cross evaluated a group recommender system with another one that would act as the ground truth source.

Generating the user groups, on the other hand, is possible. Let us now explore methods that allow us to do that.

5.3.1 Methods

Generally, we have three main methods how for creating synthetic groups:

- **At random**
- **Based on user similarity, using user-item interactions**.
- **Based on user similarity, using user-attributes**.

Creating groups at random is simple. For each group, we take the desired amount of users from the user pool without replacement. And then for the next group, either adding the users back to the pool or keeping them out and therefore having each user be part of at most one group. We could argue that entirely random groups do not exist in the real world, but some groups actually can be pretty close to random, at least in a specific preference, such as music taste. For example, when commuting on public transportation or dining in a restaurant, we would observe a very wide spectrum of preferences among people.

The second and third cases are more interesting because there is only rarely a situation where groups are created entirely randomly. Most of the time, users that create groups in the real world do have some similarities, either in preference (the second case) or in attributes, such as where they live, what is their age, gender, education, and so on (the third case).

In reality, we most usually create groups with people in the same social setting, such as age-related peers, friends from high school, university, work, and other similar social settings and attributes. It would therefore make sense to utilize this information for the creation of artificial groups. But unfortunately, this information is not present in any of our datasets, and it would need to be quite extensive in order to give us the desired outcome. This type of data is almost unattainable, maybe except for the biggest tech companies such as Google and Facebook.

Another argument against using user attributes is that when we utilize them as a grouping parameter, the concern for privacy and protection of sensitive attributes arises. These attributes should be protected as discussed in Section 3.1.

Using user-item interactions for measuring similarity is very convenient due to this information being the most accessible. It is similar to user-based collaborative filtering discussed in Subsection 2.1.2. The main differences are that we want to directly control the amount of similarity for creating either similar groups or groups with varying amounts of diversity and that we do not perform the second step, where we recommend items based on the relevant users.

In order to create user groups from user-item interactions, we need a similarity metric that gives us a value representing how similar a pair of users is. There is a multitude of similarity metrics that can be used. We have selected to use the following:

- **Pearson Correlation Coefficient** (PCC) due to its stable performance as mentioned in [57], and due to its wide use in the recommender system domain. For two vectors X and Y it can be computed as follows:

$$P(X,Y) = \frac{cov(X,Y)}{\sigma_x \sigma_y} = \frac{\sum_{i=0}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=0}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=0}^{n}(y_i - \overline{y})^2}}, \tag{5.1}$$

  where *cov* is covariance, and σ is standard deviation.

- **Jaccard similarity** due to its simplicity and suitability for data with only one-valued implicit feedback:

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}. \tag{5.2}$$

  Where X and Y are sets of items that the users interacted with, or in other words, have a positive rating of 1.

  For our case, we consider the intersection to be only when both samples have a positive value of 1. Sometimes matching zeroes is considered as an intersection as well.

- **Cosine similarity** due to its simplicity and property of not depending on the vector's magnitude, but only on their angle:

$$cos(X,Y) = \frac{X \cdot Y}{||X|| \cdot ||Y||} = \frac{\sum_{i=0}^{n} x_i y_i}{\sqrt{\sum_{i=0}^{n} x_i^2}\sqrt{\sum_{i=0}^{n} y_i^2}}, \tag{5.3}$$

  where X and Y are vectors of user preference. This insensitivity to scaling comes in handy when working with different types of ratings among multiple datasets.

We have compared the similarity metrics on a random sample of 1 million user pairs on 5.3. Initially, we wanted to select Pearson Correlation Coefficient based on the before-mentioned stability, but interestingly, PCC and the Cosine similarity overlap substantially. This is due to the ratings being very sparse, which causes the means in PCC calculations to be close to 0, which corresponds to the Cosine similarity. Further, PCC and Cosine similarity, on average, have more samples with higher similarity and fewer samples with lower similarity. We, therefore, select Cosine similarity as our similarity metric due to its similarity to PCC and better properties than Jaccard similarity.

Now, we have a method of determining the similarity between two users. Further, we need to know how to modulate the amount of similarity in the group. There are many interesting ways how to create synthetic groups. Let us now present some of them.

Some of the possible types of group creation:

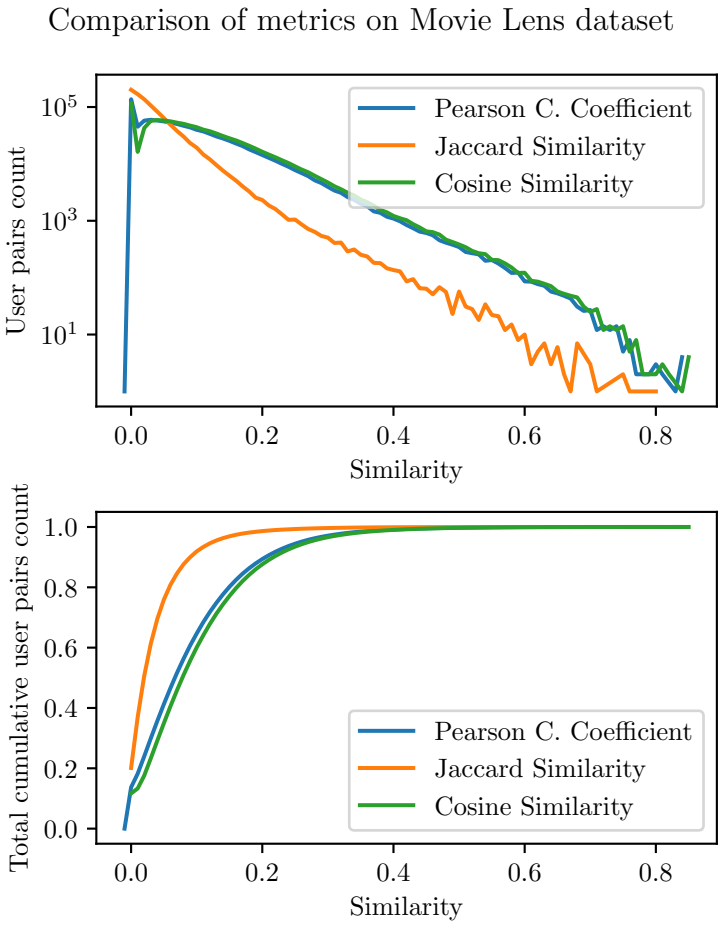


Figure 5.3: Comparison of similarity metrics on Movie Lens dataset.

- **Random:** Randomly sample without replacement from the user pool.
- **Similar:** Select one user randomly as a pivot, then fill in the remaining group spots with users that are as similar as possible. This basically is the k-nearest neighbors algorithm. The problem with this approach is that very rarely do we encounter situations like this in the real world. Another problem is that selecting the most similar users is computationally expensive as we need to compute the similarity between the pivot and every other user in the dataset, which is not ideal for the size of our data.
- **Somehow similar, with outliers:** One other way how to relax the similarity is to randomly select a pivot user and then select the nearest neighbors only from a random subset of the whole user pool. This way, we still get pretty similar users and do not need to perform so much computation.

  Another idea would be to select the next candidate based on the similarity to the last selected user instead of the pivot. In a way, we are creating a chain of users based on the similarity of the individual links(users). This way, we could create more variable preferences among the group members with still overlapping user-item interactions.

  Additionally, instead of selecting top-k similar users, we can make random steps in the similarity ordering, such as if we have 1000 most similar users

ordered by the similarity metric, we do not need to take the top-k best ones, but for example, 100th most similar, 200th most similar and so on.

Further, instead of a top-k, we can select users based on a weighted probability generated from the similarity or the ordering of all the candidate members.

- **Similar to the whole group:** Instead of selecting candidates similar to a single primary user, we can select the following group members to incrementally be similar to group members already assigned, either by aggregating the ratings or by, for example, Borda count. This has the advantage of easier candidate selection due to a bigger possible preference overlap. However, it has the disadvantage of needing to perform more similarity computations after each member selection.

- **Dissimilar:** Apart from finding group members based on being as similar as possible, we can also try to create as dissimilar groups as possible. Here we can randomly select a first user and then select a following user, which is as little similar as possible. Next, instead of selecting another user in the same way, we can calculate the dissimilarity of all other uses with the joined preference of the users already selected for the group and, again, select the most dissimilar one.

5.3.2 Selected approach

Let us now specify, in detail, the methods that we use and how we set the required parameters.

The simplest and most convenient option would be to calculate the similarity matrix for all users and then select groups based on this single calculation. However, this matrix can become pretty huge with the growing number of users. Let us assume our worst case of 1,000,000 users of the Spotify dataset. Then we will have $10^{12}/2$ similarity calculations that need to be processed and stored. This becomes difficult to manage as the amount of memory needed would be around the order of terabytes and the time needed for processing on the order of hours or days. We, therefore, opt for an iterative approach, where we randomly select only a smaller subset of potential group members and select the actual group members from this smaller group instead of the whole user pool. This can lead to calculating the users' similarity multiple times, but that situation is somewhat unlikely, and in total, we will save orders of magnitude worth of similarity calculation (potentially depending on the number of groups we want to generate).

At random

A straightforward approach that generates groups with very distinct user preferences. We select the desired amount of users for one group from the user pool and assign them together. Then repeat with the whole user pool (not removing the selected members) again.

Similar groups with top-k selection

We select one user as a pivot of the group at random. Next, we select some number of users randomly from the dataset as candidates and calculate the similarity between the pivot and each of the selected user candidates. Next, we select the top-N most similar users to the pivot where N is the desired group size - 1 and fill in the rest of the group with these top-N most similar.

This ensures that there will be pressure towards similarity using the top-K mechanism as well as some variety due to the random selection of candidates.

Similar groups with probability respecting similarity (PRS)

We perform the same random selection as in the previous method for similar groups but with a different procedure in the second step of selection from the candidates. We want to select a candidate with a probability that corresponds to their similarity with the pivot user and decreases quite heavily to ensure that there is still enough push toward selecting similar users.

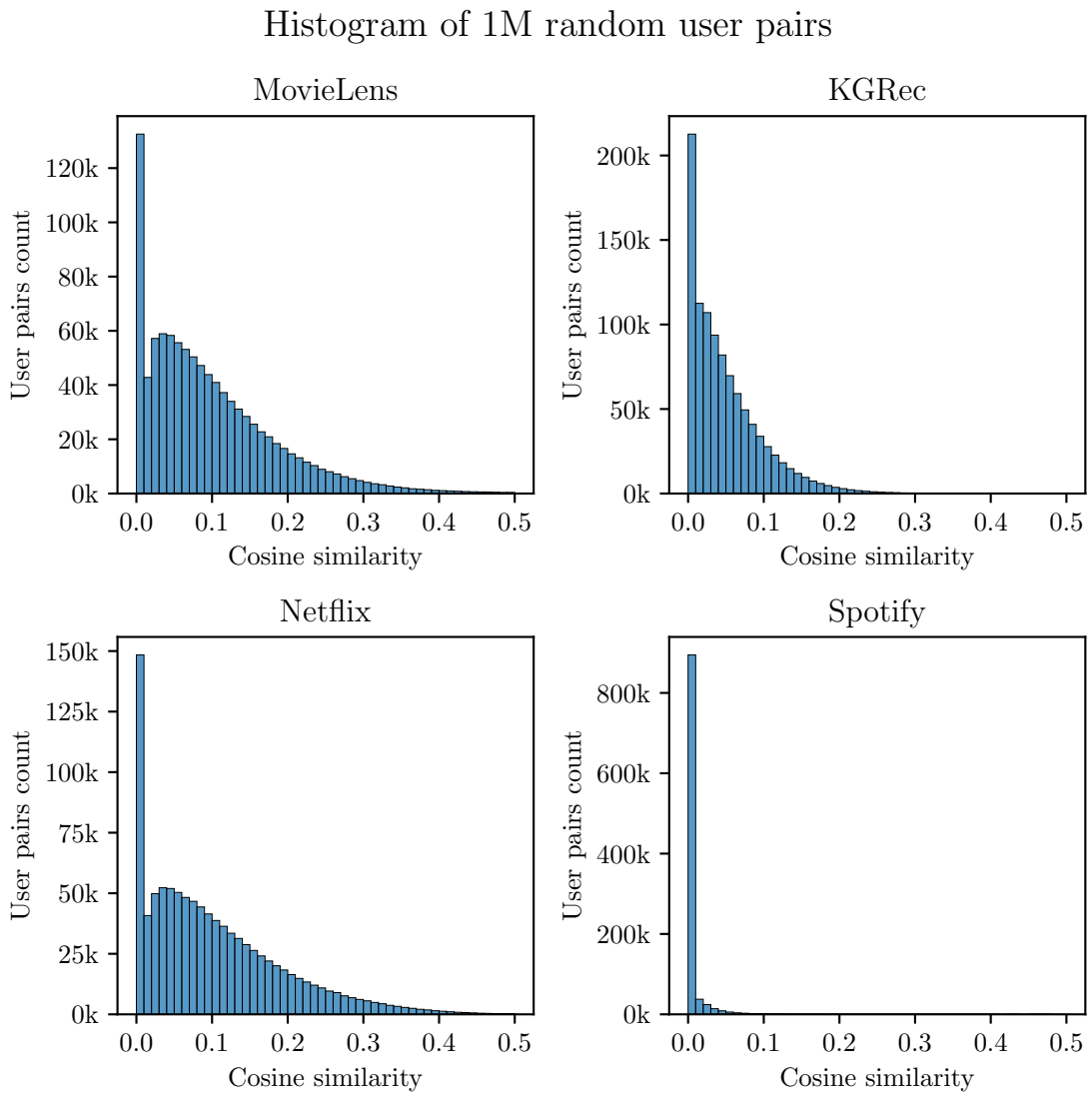


Figure 5.4: Histogram of cosine similarity of 1 million random user pairs from Movie Lens dataset.

Due to the exponentially decreasing number of users with higher similarity scores, as can be seen in Figure 5.3 and detail in Figure 5.4, we need to ensure that potential group members with higher similarity still retain overall higher probability of being selected. We can do that by presetting the desired probability to each cosine similarity group and then dividing this probability by the average number of user pairs in that similarity group.

Unfortunately, we do not have the precise average number of user pairs, which would require us to compute the similarity between all possible pairs. We can get a good estimation by sampling some user pairs and calculating the ratio between the size of the group and the number of total samples. Sample of size 1 million on all datasets can be seen in Figure 5.4.

In order to avoid deciding on the correct amount of bins to split the data into and how to smooth and sample from the bins, we have selected to model the expected cosine similarity probability density function minimizing the L2 norm. That allows us then to calculate what is the expected amount of samples in a small neighborhood. For Movie Lens, we have selected an exponential distribution with parameter $\lambda = 0.08918$. This distribution models our underlying data quite well. Comparison can be found in Table 5.10. A most important portion of samples between 0.4 to 0.8, where most of the members with the highest chance of being selected will fall, seems to be modeled quite precisely. Then we can express the average expected amount of similarity pairs with similarity of x in some small constant-sized neighborhood of size NS to be:

$$expectedSRI^2(x, N, \lambda) = CDF(exp(\lambda), x + N) - CDF(exp(\lambda), x), \qquad (5.4)$$

where CDF stands for the cumulative distribution function of exponential distribution with the λ parameter of 0.08918.

cosine sim.	sampled data	$exp(\lambda = 0.08918)$
$[0.0, 0.1)$	600,716	674,139
$[0.1, 0.2)$	275,099	219,675
$[0.2, 0.3)$	90,355	71,583
$[0.3, 0.4)$	24,265	23,326
$[0.4, 0.5)$	6,660	7,601
$[0.5, 0.6)$	2,181	2,476
$[0.6, 0.7)$	602	807
$[0.7, 0.8)$	110	263
$[0.8, 0.9)$	12	85
$[0.9, 1.0]$	0	27
d total	1,000,000	999,982

Table 5.10: Comparison of histograms of data sampled from the dataset and the created data model.

However, we are not necessarily interested in the expected amount of samples on an interval. The main requirement is to have a ratio of expected occurrence between the sampled similarities. We are not interested in the actual value. Therefore approach using a probability density function of the modeled distribution is better. We can calculate the ratio as follows:

[2]expected sample ratio on interval

$$expectedSampleRatio(x, \lambda) = pdf(exp(\lambda), x) = \lambda e^{-\lambda x}, \tag{5.5}$$

where pdf represents the probability density function.

Next, we need to know how to scale the actual similarity. We would like to have a non-linear difference between samples. If sample A has a similarity higher than sample B by 0.1, then we would like to have sample A be selected twice as likely. Further, we can generalize the difference as parameter D and the multiplying factor as M.

We get:

$$scale(x, D, M) = M^{\frac{1}{D}x} = M^{\frac{x}{D}}, \tag{5.6}$$

where for doubling the probability, every 0.1 similarity difference will become:

$$scale(x, 2, 0.1) = 2^{\frac{x}{0.1}} = 2^{10x}. \tag{5.7}$$

$$\begin{aligned} weight(x, D, M, \lambda) &= \frac{scale(x, D, M)}{expectedSampleRatio(x, \lambda)} \\ &= \frac{M^{\frac{x}{D}}}{\lambda e^{-\lambda x}} \end{aligned} \tag{5.8}$$

Parameter D in scale calculation has a significant impact on scaling. Unfortunately, it needs to be tuned to each dataset separately due to the different rating distributions, as shown in Figure 5.4. Through experimentation, we have observed that the value of 0.1 seems to give us good results on every dataset apart for Spotify, where the width needs to be shortened substantially. We solve this situation by setting the parameter to the width of the interquartile range of the sampled ratings. In other words, we set it to the width of the middle 50% mass for each dataset. We can see the resulting parameters for each dataset based on 1 million random user pairs in Table 5.11.

dataset	interquartile width (D)
MovieLens	0.104
KGRec	0.057
Netflix	0.114
Spotify	0.025

Table 5.11: Scaling width parameter based on the interquartile range of each dataset.

This gives us a framework for assigning each candidate member a weight by which we can perform a weighted random choice and therefore introduce more variability into the group selection process. The entire equation for a sample's weight can be calculated with Equation 5.8. Further, we describe the whole group generation process in Algorithm 1.

5.3.3 Evaluation of the generated groups

We have implemented and used the selected techniques to generate synthetic groups. Let us now evaluate the performance of each method. Next, we discuss the parameters of the generation process.

Algorithm 1 Generate groups with probability respecting similarity

Select scaling parameter M and the desired amount of samples S and amount of candidates C

for Desired number of samples S **do**
    Select random pair for users $u1$ and $u2$ from the user-pool
    Calculate similarity of $u1$ and $u2$ and store it
end for
Fit an exponential distribution to the stored values
Calculate the width between 25th and 75th quartile as parameter D
Get its parameter λ

for Number of desired groups **do**
    Select 1 user *uCaptain* from the user-pool as main user
    Select C random users as *uCandidates* from the user-pool
    for Each user u from *uCandidates* **do**
        Calculate similarity between *uCaptain* and u as *sim*
        Calculate random candidate weight with (Eq. 5.8):
        weight = scale(*sim*, D, M) / expectedSampleRatio(*sim*, λ)
    end for
    for Desired group size - 1 **do**
        Perform weighted random selection on the candidates using the weights
        Add the selected user to the group and remove it from the candidates
    end for
end for

The random method is the simplest from the point of selecting parameters of the group creation. It does not have any.

Next, the top-k method, it only has a single parameter of the number of candidates from which we choose the actual group members. We have set the default number of candidates to 1000 due to the computational complexity rising linearly with the number of candidates.

Lastly, our PRS method. It has three parameters in total, where λ is the parameter of the exponential distribution fitted to a sample of user similarities. D is the distance, and M is the multiplier factor of the scale ratio. We have set the distance to be 0.1, and we vary the multiplier factor from 0,5 to 4.

We aim to compare the mean inter-group similarity. First, we define a subset of size n of set S as

$$R(S, n) := x \in \mathcal{P}(S) : |x| = n, \tag{5.9}$$

where $\mathcal{P}$ is a powerset, a set of all subsets.

Then we define the inter-group similarity, which represents the mean of similarity between all tuples in the group as

$$intergroupSimilarity(G) = \frac{1}{|R(G,2)|} \sum_{\{a,b\} \in R(G,2)} cos(a, b). \tag{5.10}$$

With each group generation method, we have created 1000 groups of size 5. The resulting mean inter-group similarities can be seen in Figure 5.5. As expected, the random and top-k have the least and most mean inter-group similarity. With PRS, we have achieved the desired scaling of the mean inter-group similarity between the random and top-k. Furthermore, we see that PRS generates groups that are comparably or more varied in the similarity as top-. Other methods, which manage to create groups that utilize some similarity scaling, such as in [45] that select candidates based on their similarity with the pivot will not produce groups that vary in the inter-group similarity.

At first, PRS did not scale well on the Spotify dataset, with parameter D set to 0.1. The dataset has its similarity concentrated closer to zero. This is because most users (playlists) did not rate more than 30 items. The majority of ratings are between 0 and 0.1; therefore, we need to set the scaling width to a lower value to compensate for this. Based on this, we have introduced the technique of setting the D parameter to the width of the interquartile range, which solves the issue and decreases the number of parameters that need to be tuned to only one - the scaling parameter M.

In conclusion, we are satisfied with the PRS technique. It provides us with a method of reliable generation of groups of varying inter-group similarity, which are more heterogenous and therefore closer to reality than selecting group members solely based on the desired similarity.

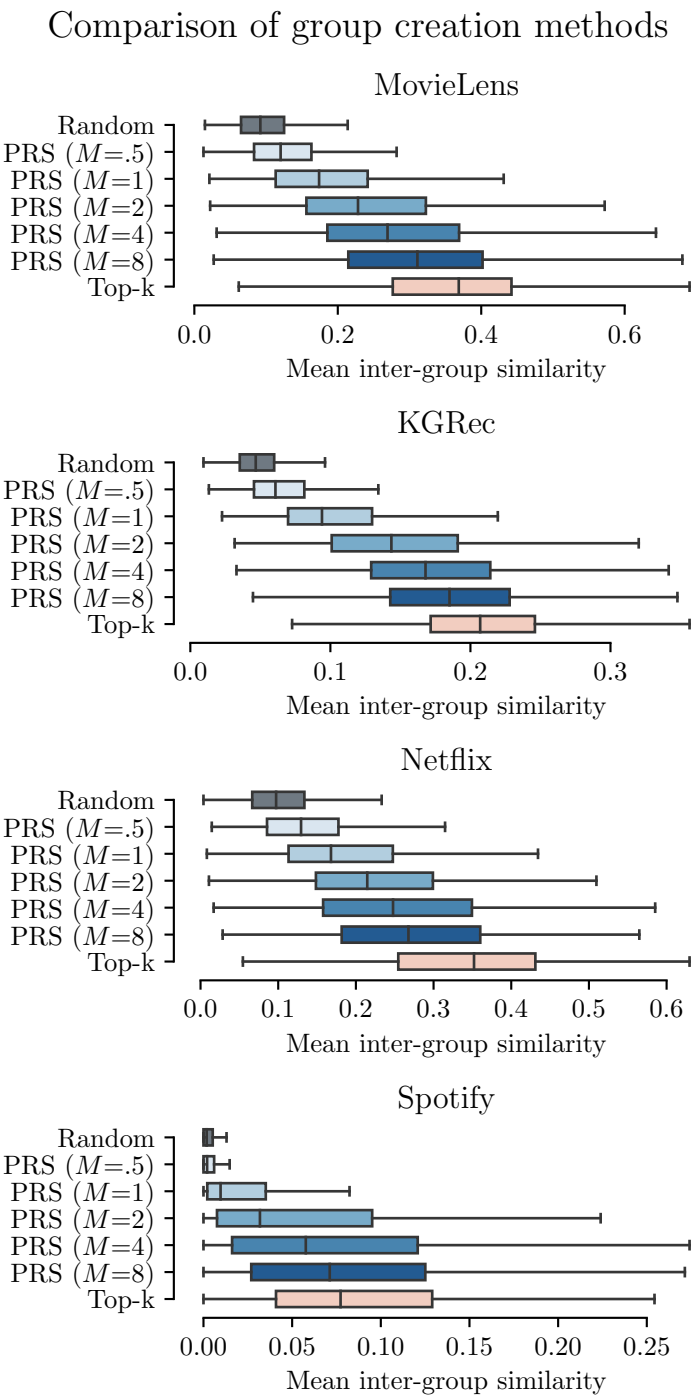


Figure 5.5: Comparison of Random, Top-k and multiple variants of PRS group generation for group size of 5. The number of candidates for PRS and Top-k was 1000.

6. Proposed group recommender system

As we have discussed so far, group recommendation is not an easy task. There are many social and mathematical optimization criteria that are hard to define and even harder to measure. Let us now present our algorithm that aims to provide group recommendations that optimize fairness and relevance. We published this algorithm originally at UMAP'21 in [58].

6.1 Preliminary

In our context, we see fairness as a property that no user should be systematically discriminated against when it comes to their satisfaction with the recommended items. Optimizing this type of fairness is hard, especially when we have a user or a subset of the group whose preferences differ from the rest.

Some algorithms, such as the Average (AVG) and Least Misery(LM), already optimize for fairness (even if not primarily designed for that purpose). We can say that AVG strategy is fair due to its property that it considers all members' preferences as equally important. If the task is to recommend only a single item, then there is not much improvement to be made apart from how we define what we consider fair. However, that is different for tasks where we recommend a list of items or we have multiple following recommendation sessions. Then we can tweak our following recommendations to fix our previously introduced fairness imbalance.

Let us now classify approaches by how they optimize fairness into the following groups: *item-wise*, *list-wise*, and *rank-sensitive* fairness.

Firstly, item-wise approaches optimize single-item performance. They usually apply some form of aggregation, such as average for the AVG or minimum for the LM approach. Scores for each item are calculated independently, and so scores for one item do not affect scores for any other.

Secondly, *list-wise* approaches take into consideration which items have been recommended before. This allows us to balance fairness among group members over multiple consecutively recommended items. All before mentioned methods from Subsection 3.2.1 fall into this category. In other words, items are not recommended independently, but the total fairness of items recommended in succession is important. The simplest example of a *list-wise fairness* approach is FAI. We select for each user, in turn, their most preferred item. This way, at some point, each user will be satisfied, and FAI therefore indirectly balances the per-user utility.

We can define a per-user utility function $r_u; u \in G$, which measures the specified properties that we aim to optimize during the process of generating a list of recommended items. We can then use this utility function to control the recommendation of the following items to balance fairness among the group members. Most commonly, the utility function is a mean of predicted relevance for each user.

Moreover, lastly, *rank-sensitive* approaches optimize fairness for each prefix of

the recommended list of items. This approach has theoretically ideal properties in use cases where we do not know what portion of the recommended list will be consumed or viewed by the users. We only know about a single related approach from this category, GFAR [45]. As already described in Subsection 4.3.1, the authors define fairness through the sum of probabilities that at least one item from the list will be relevant to each user.

Our proposed algorithm falls into the group of *rank-sensitive* approaches. It maximizes the per-user proportional fraction of total relevance. Compared to GFAR, it does not define fairness based on the ranks of items. This gives it more freedom in the item selection process due to rank sensitive approach discounting the relevance of items with decreasing rank quite quickly, even if, in reality, items on lower ranks can still present a great option.

As already mentioned in our paper about this algorithm [58], the main inspiration for our work is election algorithms that are used for mandates allocation. Mandate allocation is an important problem in distributing mandates based on the votes received. The important part of the problem is what to do if the votes are not divisible in the way that would allow to distribute them by a simple one mandate per a set number of votes without any remaining votes left unassigned.

We focus on D'Hondt's algorithm (DA). It, as already stated in Subsection 4.3.3, is commonly used in election systems around the world. It is a greedy selection algorithm that minimizes the number of votes that need to be left aside so that the remaining votes are represented exactly proportionally as described in [48].

D'Hondt's algorithm procedure works in discrete steps, where in each step, it selects the party with the maximum quotient *quot* one seat. The quotient for each party is calculated as the total number of votes the party has received V_p divided by the number of seats $s_p + 1$ that has already been awarded to that party p. Initially, s_p is set to zero.

$$quot(V_p, s_p) = \frac{V_p}{s_p + 1}, \tag{6.1}$$

Directly using the DA algorithm is not possible due to it being designed for discrete mandate allocation problems. But this unfortunate property has been addressed in [49]. We describe the method of D'Hondt's direct optimization in more detail in Subsection 4.3.3.

6.2 EP-FuzzDA

Let us now describe our algorithm of **E**xactly **P**roportional **F**uzzy **D** 'Hondt's **A**ggregation. It is a greedy algorithm that iteratively selects the best candidate item maximizing the marginal gains on **E**xactly-**P**roportional **rel**evance **sum** (EP-rel-sum) criterion.

As mentioned before, candidate allocation algorithm such as D'Hondt does not consider the varying relevance of candidate items nor the possible overlap of user preferences. Both these situations are very common in the domain of group recommender systems. We can probably find a set of items that closely balance the relevance between all group members. Nevertheless, each of these items may have a different candidate item that is a better fit for the group in the average

rating but would lead to more imbalance. This could easily lead to recommending mediocre items only for the sake of balancing fairness.

We, therefore, want to maximize the balance of items' utility for each group member but without hindering the possible additional utility provided to members that overlap in preference with other members or are easier to satisfy. One solution is to optimize for the total item relevance but cap the maximum possible gain for each individual user so that it does not further add to the total item score. We call this cap *exactly proportional relevance allowance*. It is calculated as a proportional share of each user on the total sum of all so far selected items' relevance.

Let us assume that we have already selected items $\mathcal{L}$ (this set can be empty), with individual relevance of $r_{i,u}$ for all items $i \in I$ and users u of a group $\mathcal{G}$. We will use the example in Table 6.1 for the following calculations.

						EP-FuzzDA	
Object	u_1	u_2	u_3	AVG	LM	step 1	step 2
c_1	0.9	0.8	0.0	0.567	0.0	1.13	-
c_2	0.6	0.5	0.0	0.367	0.0	0.73	0.17
c_3	0.5	0.9	0.0	0.467	0.0	0.93	0.37
c_4	0.0	0.1	0.9	0.333	0.0	0.43	1.00
c_5	0.1	0.1	0.8	0.333	0.1	0.53	0.90
c_6	0.2	0.0	0.7	0.300	0.2	0.50	0.70

Table 6.1: Example of relevance calculation for item-wise aggregations. Least-misery, average, and two steps of our DA method. Step 2 is after selecting c_1 in step 1.

We then calculate the relevance allowance for a new candidate item as follows. First, we sum up relevance for all users for all already selected items as

$$TOT(\mathcal{G}, \mathcal{L}) = \sum_{i \in \mathcal{L}} \sum_{u \in \mathcal{G}} r_{i,u}, \tag{6.2}$$

Then we define TOT_c for a candidate item c (different from all already selected) as $TOT(\mathcal{G}, \mathcal{L} \cup c)$. This represents the prospected total utility if we would select the candidate c.

We then calculate the maximum allowed utility (relevance) with the appropriate weight of a user w_u as:

$$allowed\_utility_u = TOT_c * w_u. \tag{6.3}$$

The weight of each group member can either be set to $1/|\mathcal{G}|$ or arbitrarily so that $\sum_{u \in \mathcal{G}} w_u = 1$. This weight represents how important each user is in the group and will scale the preferred utility ratio between group members if not set uniformly.

In our example in Table 6.1, at step 1. we have $TOT_{c1} = 0.9 + 0.8 + 0.0 = 1.7$ which scaled by uniform weight of $1/3$ to $1.7 * (1/3) = 0.57 = allowed\_utility_u$. We then calculate the unfulfilled relevance as the total received utility (from already selected items in $\mathcal{L}$) subtracted from the maximum allowed utility. We are selecting the first item. Therefore all users start with a received utility of

0, which makes the unfulfilled relevance for candidate item c_1 set to 0.57 for all users.

$$total\_received\_utility_u = \sum_{i \in \mathcal{L}} r_{u,i}, \tag{6.4}$$

$$unfulfilled\_rel_u = max(0, allowed\_utility_u - total\_received_u tility_u). \tag{6.5}$$

We can then finally calculate the utility of each item by

$$rel_i = \sum_{u \in \mathcal{G}} min(rel_{u,i}, unfulfilled\_rel_u). \tag{6.6}$$

We can now calculate the utility of item $c1$ as $min(0.9, 0.57) + min(0.8, 0.57) + min(0.9, 0.57) = 1.13$. And similarly for all other candidate items. After the first round, we select $c1$ and follow into a second one. Now it starts to get interesting because we need to calculate the total received utility and use it to calculate the unfulfilled relevance correctly.

For item c_2 we have $TOT_{c_2} = 1.7 + 0.6 + 0.5 + 0.0 = 2.8$ (1.7 is TOT) and user u_1 we have unfulfilled relevance of $2.8/3 - 0.6 = 0.03$, for user u_2 we have $2.8/3 - 0.8 = 0.13$ and for u_3 we have $2.8/3 - 0.0 = 0.93$. All numbers are capped to the min 0. So in total the relevance of item c_1 is $min(0.6, 0.03) + min(0.5, 0.13) + min(0.93, 0.0) = 0.16$. We can see the output of relevance calculations for all items in the example table under column step 2. In the second step, we would therefore select item c_4.

Let us now be more formal with the definition of EP-rel-sum. We have a list of recommendations $\mathcal{L}$ and its total relevance TOT as defined previously. Next, we have the r_u total relevance of all items from $\mathcal{L}$ for user u as defined with Equation 6.4. Further, we have the per-user proportional relevance allowance a_u ($allowed\_utility_u$) as $TOT * w_u$. Weights $\sum_{u \in \mathcal{G}} w_u = 1$. Then we finally define the EP-rel-sum criterion as

$$\text{EP-rel-sum}(\mathcal{L}_G) = \sum_{\forall u \in \mathcal{G}} \min(r_u, a_u). \tag{6.7}$$

We can observe that for two lists L^1_G and L^2_G with the same total relevance, EP-rel-sum will be higher for the list more proportional to the weights w_u distribution. Another way around, if we have two lists with the same relevance distribution, the one with higher total relevance will receive a higher EP-rel-sum score.

And finally, to make EP-rel-sum ranking sensitive, as previously described, we define marginal gains while extending the list of items as

$$gain(\mathcal{L}_G, c_j) = \text{EP-rel-sum}(\mathcal{L}_G \cup \{c_j\}) - \text{EP-rel-sum}(\mathcal{L}_G). \tag{6.8}$$

EP-FuzzDA, which we describe in Algorithm 6.2, iteratively selects the item that provides the maximal marginal gain to the already selected set of items. It, therefore, is a *rank-sensitive* approach for optimizing fairness.

Another nice property not present in other group recommendation algorithms is that our algorithm naturally incorporates scaling of user preference importance using the weights w_u. We will use this later in our experiments for weighted groups and long-term fairness evaluation.

Algorithm 2 Exactly-Proportional Fuzzy D'Hondt's Aggregation

Input: group members $u \in \mathcal{G}$, candidate items $i \in \mathcal{I}$, relevance scores $r_{u,i} \in \mathbf{R}$, #items to recommend k, user's weights w_u; $\sum w_u = 1$
Output: ordered list of group recommendations $\mathcal{L}_G$ of size k
$\mathcal{L}_G = []$
$TOT = 0$
$\forall u : r_u = 0$
for 1 to k **do**
    for $i \in \mathcal{I} \setminus \mathcal{L}_G$ **do**
        $TOT_c = TOT + \sum_{\forall u} r_{u,i}$
        $\forall u : allowed\_utility_u = TOT_i * w_u$
        $\forall u : unfulfilled\_relevance_u = max(0, allowed\_utility_u - r_u)$
        $gain_i = \sum_{\forall u} min(r_{u,i}, unfulfilled\_relevance_u)$
    end for
    $i_{best} = \text{argmax}_{\forall i}(gain_i)$
    append c_{best} to $\mathcal{L}_G$
    $\forall u : r_u = r_u + r_{u,i_{best}}$
    $TOT = \sum_{\forall u} r_u$
end for
return $\mathcal{L}_G$

7. Experiments

Let us now talk about our experiments. First, we describe the evaluation process, which we use, and the rationale behind it. Then, we present the data flow and software we have written, and finally, we present the results.

The following experiments are an extension of our original research paper [58]. Differences and extensions that have been made will be discussed further. Most notably, they are the addition of 3 new datasets, different matrix factorization algorithms, and a different group generation method.

All code for Python scripts and Jupyter notebooks can be found at github.com/LadislavMalecek/MasterThesisAnalysis. All file paths mentioned in this chapter assume to have the base path at the root of this repository (when cloned on the disk).

7.1 Evaluation

Our experiments are run in an offline setting. We therefore only have the users' preferences, not the actual group ratings. Further, as discussed in Chapter 5, we do not have a dataset that would contain group preferences, nor a dataset that would contain any group information. Let us now discuss the separate parts of the evaluation and the choices we have made in the experiments. Later we will put everything together and go through each step of the experiment.

7.1.1 Coupled vs. decoupled

One of the problems with group recommendation systems is the evaluation. In the case of a single-user RS, it is common to have the data split into training, validation, and testing parts. This way, for example, if we train a matrix factorization algorithm, we can check how it is performing at the end of our training using the testing portion of the dataset which is separated from the data that has been used for training. This can be used even if the user ratings are very sparse, which in the majority of cases they are. But with multiple users, such is the case for group RS, we would need to have the ratings overlapping for the group members, meaning that they all have rated the same items. Even worse, with each additional member, the overlap of ratings will become smaller and smaller.

We can distinguish between two main approaches for evaluating group recommendation systems - coupled and decoupled.

The main difference, as we can see in Figure 7.1, is that in the **coupled** (tightly coupled with the underlying RS) evaluation approach, we evaluate our performance on a with-held set of testing data and in the **decoupled** evaluation, we consider the underlying single user recommendation system the source of our ground truth. The coupled approach, therefore, evaluates not only the group RS aggregation method but the underlying single-user RS as well.

The coupled evaluation has the undesired property of favoring group RS that tends to select the per-user best items as we have mentioned in [59].

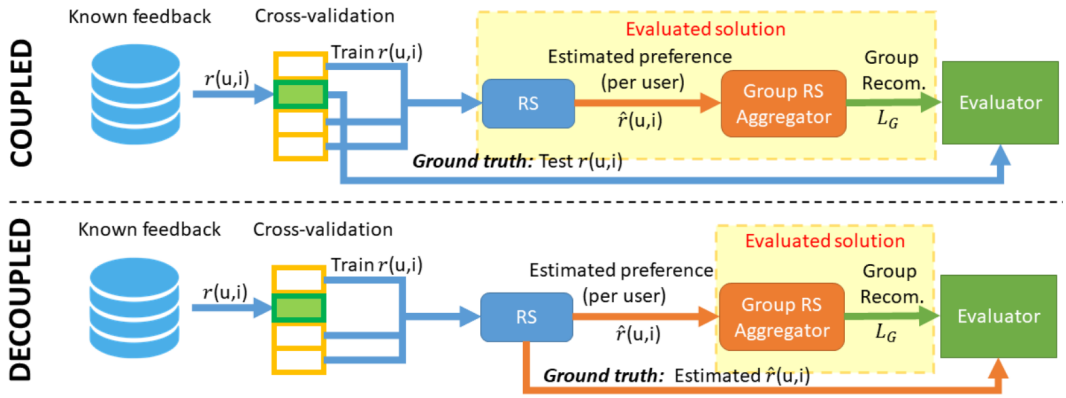


Figure 7.1: Example of two main evaluation approaches from [59].

7.1.2 Metrics

After running the group recommendation algorithm, we end up with a list of recommendations for each group. We now need to evaluate the performance of each algorithm. For each group size, group type, and algorithm, we want to get a measure of how well they have performed.

Firstly, we need to get the ground truth from our underlying RS for each item and each group member. Now for each group member and list of recommendations, we evaluate two metrics: normalized discounted cumulative gain (**nDCG**) and average relevance score (**AR**). We have chosen nDCG due to its popular use in our domain and a good representation of the probability of selection of an item based on the item's position in the list. AR metric has been chosen due to its simplicity and an overall good measure of performance for short lists of recommendations.

For each group and each user, we have **AR** and **nDCG** scores. Further, we calculate the aggregated per-group statistics for both of these metrics. We calculate **mean**, minimal user's score (**min**), and the ratio between minimal and maximal scores (**M/M**). All of these are valid fairness definitions on their own, depending on the definition of fairness as discussed in Chapter 3. These metrics are also used in related literature [45], [46]. Finally, we aggregate results for all groups together by calculating the mean of the metrics.

7.1.3 Evaluation use-cases

In [45], the authors focus only on the scenario of uniform groups, where each user is weighted uniformly. For our experiments, we are evaluating not only the **uniform** scenario but **weighted** and **long-term** scenarios as well.

In the **weighted scenario**, we randomly assign a weight to each of the group members (all summing up to a total of 1). This weight represents the non-uniform importance of each group member. This scenario is a relevant strategy for situations in structured groups where some members have a priority, such as a family with children watching a movie or a birthday party. This is an extension of the most respected person strategy described in [60], with less rigid rules of only requiring weights to be in an interval of [0,1], with the total sum of weights in the group equal to 1.

In the **long-term scenario**, we assume that, in reality, the groups are more permanent with multiple instances of recommendation that are time separated. Such as a group of friends watching a movie every week or any other fixed group that consumes content together repeatedly. In this scenario, we want to prevent a systematic bias against one or multiple members and recommend items in the subsequent sessions with respect to the satisfaction and fairness of the previous sessions. We use the weights as in the previous weighted scenario and adjust them accordingly to satisfy the under-represented uses. We define the weights as a non-negative difference between the exactly proportional share of total relevance and the sum of relevance scores that the user has gained so far.

To evaluate the long-term scenario, we recommend items five times in a row, where after each row we calculate the appropriate weights and exclude the previously recommended items from further recommendation. After these five recommendation rows, we evaluate the mean utility for each user.

7.2 Evaluation setup

The experimental setup is an extended version from our [58]. In the paper, we have used experiments from [45], that we have extended with our algorithms and weighted and long-term recommendations.

Unfortunately, the original code from [45] we have worked with is quite proprietary and written in Java. For this work, we would have to extend it even further (compared to our work in [58]) by adding the new group generation process that we have designed in Section 5.3, new datasets, and a new matrix factorization. For convenience and further future research reusability, we have therefore made a decision to write all parts of the experiment in Python.

The main notable change from [45] is that we have modified the evaluation procedure. We use the decoupled evaluation instead of the coupled one as it gives an unfair advantage to algorithms that select the per-user best items and does not evaluate the underlying single-user RS in a coupled way as mentioned in Subsection 7.1.1.

Our experiments have 5 following steps.

7.2.1 Data retrieval and processing

Unfortunately, as discussed in detail in Chapter 5, we cannot host or append the used datasets directly to our code repository due to licensing. We have to therefore start with retrieving and processing the datasets.

Run `./gather_datasets/downlad_and_transform.py` script directly if you wish to retrieve the datasets manually. It supports Movie Lens, Netflix, KGrec, and Spotify datasets and performs cleanup and transformation to make further processing of the datasets as convenient as possible.

The cleaning process transforms the datasets to a unified format with a single table of user ids, item ids, and ratings (if provided), normalizes and removes empty gaps from user and item ids, and validates the downloaded data.

We have committed the weights that are outputted from the next step of matrix factorization to the repository. Therefore it is possible to skip this step if desired.

7.2.2 Training matrix factorization algorithm

We use two matrix factorization (MF) methods to generate our ground truths for the decoupled evaluation. First is our custom implementation of the highly parallel alternating least squares method (ALS) that we use for datasets with explicit ratings. The second is an implicit variation of the ALS method from a Python library Implicit [1] that we use for datasets with implicit ratings.

As an output of this step, we get the two factors of MF, user and item factor matrix. We avoid saving the full target matrix due to the substantial size of the larger datasets, which would further increase our memory requirements. Instead, we calculate the ratings on the fly in the next steps.

Run `./matrix_factorization/matrix_factorization.py` if you wish to calculate the matrix factorization manually.

In our experiments, we use the following parameters for matrix factorization: number of iterations 15, random seed 42, regularization factor of 0.1, convergence threshold of 0.0001, and number of factors 50 for KGRec, 200 for Movie Lens, and 300 for Netflix and Spotify datasets. For KGRec and MovieLens datasets, we have selected the parameters similarly as in [45]. For other datasets, Netflix and Spotify, we have selected the parameters by hand based on a simple heuristic - the bigger and more complex the dataset, the higher the number of factors.

Matrix factorization methods tend to degenerate when selecting the wrong parameters for the size of the factors, but the results for different types of groups (based on similarity PRS(M=1) and PRS(M=4)) do suggest that this is not the case, due to better results for groups with higher similarity (groups of more similar members are easier to satisfy). If the factors would degenerate and provide us only with flat ratings, then there would not be a difference in results for the two mentioned types of groups.

We have committed the generated weights for all datasets into the code repository. Therefore it is possible to skip this step if desired. They are stored in a binary Numpy format with one exception for the Spotify dataset for which the item factor has reached the limit of GitHub LFS maximum file size. Therefore, we have zipped this dataset's factorization data in a standard .zip format.

7.2.3 Creation of synthetic user groups

We have implemented 3 group creation strategies as already discussed in Section 5.3. These are Random, Top-K, and Probability respecting similarity (PRS) algorithms. As shown in Figure 5.5, our method PRS nicely scales between the two extremes of Random and Top-K selection. We further assume that these two extremes do not often appear in real-world scenarios. Therefore, we have omitted them from our evaluation to simplify our experiments and only use groups created by the PRS algorithm.

Run `./create_groups/create_random_groups.py` to create random groups
or run `./create_groups/create_topk_groups.py` to create Top-K groups
or run `./create_groups/create_prs_groups.py` if you wish to create PRS groups.

[1] https://github.com/benfred/implicit

We use PRS with two different settings of M, first at $M = 1$ and second at $M = 4$. Based on our analysis in Subsection 5.3.3, we believe these two variants nicely represent the usual group setting.

Further, for each setting, we generate 1000 groups in sizes 2, 3, 4, 6, and 8. We use 1000 candidates for each group, and lastly, we use 10,000 samples to get the dataset's similarity distribution.

For weighted experiments, we additionally generate group weights as described in Subsection 7.1.3.

Run `./create_groups/create_group_weights.py` if you wish to generate them manually.

7.2.4 Running GRS algorithms

After the previous steps, we finally have all the required data to run GRS algorithm. We use the following algorithms from Chapter 4: **AVG**, **LM**, **FAI**, **XPO**[**46**], **NPO**, **GFAR**[**45**] together with **EP-FuzzDA** and is direct optimization variant **DHondtDO**.

We use a computation optimization for all algorithms, where for each group member, we select their top 1000 candidate items (by rating), and then we compute the algorithm's item selection only on a joined set of these top items. Note that it is 1000 for each group member, so the number of candidate items in the set for the whole group will be bigger. For XPO and NPO, we use 30 candidate items instead of 1000 due to the computational complexity of those algorithms (this will be further discussed in Section 7.4).

Complete overview of all parameters for each algorithm:

- **AVG, LM, FAI**: candidate items = 1000
- **XPO, NPO**: candidate items = 30, monte Carlo trials = 100
- **GFAR**: candidate items = 1000, max relevant items = 100
- **EP-FuzzDA, DHondtDO**: candidate items = 1000,

For each group and each algorithm, we get the top 10 items.

Run `./experiments/run_algorithms.py`, `./experiments/run_weighted_algorithms.py` and `./experiments/run_longterm_algorithms.py` to generate uniform, weighted and long-term experiments, respectively.

7.2.5 Evaluation

The last step is an evaluation of the results. We have 4 datasets, 5 group sizes of 2, 3, 4, 6, 8. 2 types of groups, and 8 GRS algorithms. and 6 metrics combination as described in Subsection 7.1.2. We evaluate these results in the following Jupyter notebooks under the directory `./evaluation/` directory. For uniform run `evaluation_uniform.ipynb`, for weighted run `evaluation_weighted.ipynb` and for long-term `evaluation_longterm.ipynb`. These notebooks are meant to be run manually to get the results, but support an execution in the style of the previous scripts by running `jupyter run ./evaluation/evaluation_uniform.ipunb`.

7.3 Results

Due to the high number of datasets, we will present the efficacy on the Movie Lens (25M) dataset. Significant results of other datasets that differ from the Movie Lens dataset will be also discussed. Results for other datasets can be found in Appendix A.

7.3.1 Uniform scenario

In Table 7.1, we can see that EP-FuzzDA performs well across all experimental metrics and group sizes. For AR metrics, the performance is mostly led by AVG and LM algorithms, where AVG dominates in AR mean, which is explainable due to its direct optimization of the highest average relevance. For AR min and M/M the performance is led by LM algorithm, with our EP-FuzzDA following with only a slight decrease in the performance and for smaller group sizes of 2 and 3 actually outperforming the LM algorithm. The slight edge of LM in these two metrics can be again explained by its direct optimization of the highest minimal relevance. Overall, we can say that our algorithm performs the best when evaluating the combination of mean, min, and M/M. It provides better mean average relevance values than LM and better min, M/M performance with only a slight decrease in mean AR compared to AVG.

For nDCG metrics, where the position of the items becomes important, it performs worse than the related DHondtDirect optimization, which performs more of a greedy recommendation strategy. Nevertheless, still, the performance is comparable to the best results.

For groups with more similarity (PRS, M=4) the results are comparable, with a slight improvement in performance for bigger but more similar groups. It is easier to recommend universally liked items when the preferences align more between the group members. For those situations, our algorithm's performance increases for AR min metric.

We conclude that, for the uniform evaluation scenario and the Movie Lens dataset, our EP-FuzzDA algorithm provides the highest fairness (in the context of our metrics) while maintaining the high relevance of the recommendation as measured by the mean AR and nDCG metrics.

Additionally, for the KGRec dataset (Table A.1), it provides recommendations with the best minimal and highest min/max score ratios for group sizes 4 and 6 and close to the best for the challenging group size 8. In more difficult scenarios of big and less similar groups $PRS(M = 1)$ group size s=8, the algorithm performs well and is only dominated in those metrics by more specialized but naive algorithms (AVG, LM).

For the smaller MovieLens1M dataset (Table A.2), and the Netflix dataset (Table A.3), the results are analogical to the Movie Lens dataset.

And finally, for the Spotify dataset (Table A.4), the results are slightly improved and are analogical to the KGRec dataset. This is interesting as it shows that for implicit datasets, the algorithm's performance seems to be better even for different sizes of the dataset (KGRec being the smallest and Spotify being the biggest).

	PRS(M=1), group size s=2						PRS(M=4), group size s=2					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**64.75**	49.19	0.67	0.85	0.78	0.85	**66.11**	50.50	0.69	0.85	0.78	0.86
FAI	56.05	48.45	0.79	0.76	0.71	0.88	57.16	49.52	0.79	0.76	0.71	0.88
LM	58.55	55.68	0.91	0.81	0.72	0.81	60.02	57.21	0.92	0.82	0.73	0.82
XPO	55.85	48.01	0.78	0.75	0.69	0.85	57.05	48.98	0.78	0.75	0.69	0.86
NPO	54.52	46.12	0.77	0.73	0.67	0.84	55.67	47.36	0.77	0.73	0.67	0.84
GFAR	48.55	43.38	*0.82*	0.68	0.63	*0.86*	49.43	44.14	*0.83*	0.68	0.63	*0.87*
DHondtDO	64.06	*52.47*	0.74	**0.86**	**0.83**	**0.94**	65.43	*54.08*	0.75	**0.86**	**0.83**	**0.94**
EP-FuzzDA	*59.61*	**57.00**	**0.92**	*0.83*	*0.76*	0.83	*61.10*	**58.56**	**0.93**	*0.83*	*0.76*	0.84

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**61.01**	42.67	0.56	0.78	0.70	0.81	**61.30**	44.04	0.58	0.78	0.71	0.81
FAI	49.62	41.07	0.72	0.67	0.56	0.74	50.03	41.94	0.73	0.67	0.56	0.74
LM	53.67	49.61	**0.86**	0.74	0.61	0.72	54.67	50.80	**0.87**	0.74	0.63	0.73
XPO	49.88	40.81	0.71	0.67	0.56	0.73	50.35	41.73	0.72	0.67	0.57	0.74
NPO	47.95	38.68	0.69	0.64	0.53	0.70	48.49	39.33	0.70	0.65	0.53	0.71
GFAR	46.78	39.83	*0.75*	0.65	0.55	0.74	47.27	40.56	*0.76*	0.65	0.55	0.75
DHondtDO	60.16	*46.00*	0.64	**0.79**	**0.75**	**0.90**	60.47	*47.25*	0.66	**0.79**	**0.74**	**0.90**
EP-FuzzDA	*56.63*	**50.21**	0.82	*0.77*	*0.67*	*0.78*	*57.29*	**51.76**	0.85	*0.77*	*0.67*	*0.79*

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**57.48**	39.52	0.51	0.75	0.66	0.78	**58.04**	40.81	0.54	0.74	0.65	0.78
FAI	45.73	36.95	0.67	0.62	0.49	0.65	46.21	37.44	0.68	0.62	0.48	0.65
LM	50.49	**45.98**	**0.82**	0.70	0.55	0.66	51.37	47.03	**0.83**	0.70	0.56	0.68
XPO	46.57	37.09	0.66	0.63	0.49	0.65	46.97	37.48	0.67	0.63	0.49	0.66
NPO	44.52	34.98	0.64	0.61	0.46	0.61	44.69	35.09	0.65	0.60	0.45	0.62
GFAR	46.06	37.69	*0.69*	0.64	0.50	0.65	46.34	38.11	*0.70*	0.63	0.49	0.65
DHondtDO	56.67	*42.49*	0.59	**0.75**	**0.70**	**0.86**	57.21	*43.78*	0.62	**0.74**	**0.68**	**0.85**
EP-FuzzDA	*53.73*	45.89	0.77	*0.74*	*0.62*	*0.73*	*54.71*	**47.54**	0.79	*0.74*	*0.62*	*0.74*

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**54.10**	36.80	0.46	0.70	0.59	0.73	**54.69**	37.94	0.49	0.70	0.59	0.73
FAI	42.04	32.27	0.61	0.57	0.39	0.53	42.47	32.62	0.61	0.57	0.39	0.54
LM	47.47	**42.51**	**0.78**	0.66	0.48	0.59	48.16	43.32	**0.79**	0.65	0.49	0.61
XPO	43.63	32.52	0.58	0.59	0.40	0.53	44.25	32.95	0.58	0.59	0.40	0.53
NPO	41.00	30.58	0.58	0.56	0.37	0.51	41.40	30.57	0.58	0.56	0.37	0.51
GFAR	44.16	33.86	*0.61*	0.61	0.40	0.53	44.54	34.32	*0.62*	0.60	0.41	0.54
DHondtDO	53.27	*39.20*	0.54	**0.71**	**0.61**	**0.77**	53.87	*40.09*	0.57	**0.70**	**0.61**	**0.77**
EP-FuzzDA	*51.03*	42.20	0.70	*0.70*	*0.54*	*0.65*	*51.84*	**43.32**	0.72	*0.70*	*0.55*	*0.67*

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**51.76**	35.51	0.45	*0.68*	0.54	0.68	**52.13**	36.38	0.48	0.68	0.55	0.68
FAI	39.89	29.15	0.56	0.55	0.32	0.45	40.11	29.52	0.57	0.55	0.34	0.46
LM	45.54	**40.37**	**0.77**	0.63	0.43	0.54	45.95	**41.19**	**0.79**	0.63	0.45	0.56
XPO	41.86	29.48	0.53	0.57	0.34	0.45	42.22	30.09	0.55	0.57	0.35	0.47
NPO	39.48	27.72	0.53	0.54	0.31	0.43	39.42	27.77	0.54	0.54	0.32	0.44
GFAR	42.17	31.37	*0.58*	0.58	0.35	0.46	42.50	31.69	*0.59*	0.58	0.37	0.49
DHondtDO	51.10	*37.59*	0.53	**0.68**	**0.56**	**0.70**	51.49	*38.31*	0.56	**0.68**	**0.56**	**0.71**
EP-FuzzDA	*49.34*	40.13	0.67	0.68	*0.49*	*0.60*	*49.91*	40.95	0.69	*0.68*	*0.51*	*0.63*

Table 7.1: Results of offline **uniform** evaluation on **MovieLens25M** dataset. The best results are in bold, the second-best are underscored, and the third-best results are in italic.

7.3.2 Weighted scenario

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**61.01**	0.01	0.17	**0.78**	0.01	0.15	**61.30**	-0.02	0.17	**0.78**	-0.02	0.15
AVG	58.30	0.46	0.13	0.75	0.46	0.11	58.47	0.45	0.13	0.76	0.45	0.11
DHondtDO	57.84	0.46	0.13	0.76	0.46	0.11	58.29	0.45	0.13	0.76	0.45	0.11
EP-FuzzDA	54.74	**0.73**	**0.10**	0.74	**0.73**	**0.10**	55.30	**0.73**	**0.10**	0.74	**0.73**	**0.10**

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**57.48**	0.00	0.13	**0.75**	0.00	0.11	**58.04**	-0.01	0.13	**0.74**	-0.01	0.12
AVG	55.13	0.43	0.11	0.72	0.43	0.08	55.39	0.46	0.10	0.71	0.46	0.08
DHondtDO	54.79	0.45	0.10	0.73	0.45	0.09	54.89	0.47	0.10	0.72	0.47	0.09
EP-FuzzDA	52.45	**0.65**	**0.08**	0.72	**0.65**	**0.08**	53.16	**0.65**	**0.08**	0.71	**0.65**	**0.08**

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**54.10**	0.02	0.09	**0.70**	0.02	0.08	**54.69**	-0.02	0.09	**0.70**	-0.02	0.08
AVG	51.93	0.40	0.07	0.68	0.40	0.06	52.50	0.41	0.07	0.68	0.41	0.06
DHondtDO	51.52	0.42	0.07	0.68	0.42	0.06	52.16	0.43	0.07	0.68	0.43	0.06
EP-FuzzDA	49.89	**0.58**	**0.06**	0.68	**0.58**	**0.06**	50.53	**0.58**	**0.06**	0.67	**0.58**	**0.06**

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**51.76**	0.01	0.07	**0.68**	0.01	0.06	**52.13**	0.02	0.07	**0.68**	0.02	0.06
AVG	49.83	0.40	0.06	0.65	0.40	**0.05**	50.28	0.41	0.06	0.66	0.41	0.05
DHondtDO	49.52	0.42	0.05	0.66	0.42	0.05	49.92	0.43	0.05	0.66	0.43	0.05
EP-FuzzDA	48.33	**0.53**	**0.05**	0.66	**0.53**	0.05	48.82	**0.55**	**0.05**	0.66	**0.55**	**0.05**

Table 7.2: Results of offline **weighted** evaluation on **MovieLens25M** dataset. The best results are in bold. The second-best are underscored.

In Table 7.2, we see results for the weighted scenario. Our algorithm performs substantially better than AVG and DHondtDO for correlation and mean absolute error (MAE) metrics. These metrics tell us the correlation and MAE between the AR and nDCG for the group and the actual weight that has been randomly generated. We see that DHondtDO does a substantially worse job at weighing the group member preferences for a slight increase in the mean relevance performance compared to our algorithm. This is understandable. The act of weighing the relevance among the group members has to have some impact to the average performance. We have added the AVG-U algorithm, which is an AVG, but with the weights set to uniform as in the previous uniform scenario. We have added the AVG-U to serve as an upper bound for mean and a lower bound for correlation and MAE metrics. It has the best mean relevance performance but without the results correlating with the group weights.

Results for the KGRec dataset (Table A.5), the MovieLens1M dataset (Table A.6), the Netflix dataset (Table A.7), and the Spotify dataset (Table A.8) are analogical to results for the MovieLens25M dataset.

7.3.3 Long-term scenario

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	281.0	206.9	0.60	2.09	1.94	**0.87**	282.4	213.0	0.62	2.09	1.95	**0.87**
AVG	264.6	233.3	0.80	2.07	1.88	0.84	268.7	239.0	0.81	2.08	1.90	0.85
DHondtDO	**283.3**	**246.7**	0.79	**2.18**	**1.95**	0.83	**287.8**	**253.5**	0.80	**2.19**	**1.98**	0.84
EP-FuzzDA	259.9	241.2	**0.89**	2.05	1.71	0.73	263.4	247.7	**0.90**	2.05	1.73	0.74

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	266.7	194.0	0.55	2.01	**1.83**	**0.84**	268.7	200.2	0.58	1.98	**1.81**	**0.84**
AVG	252.8	217.2	0.76	2.00	1.74	0.79	256.7	223.6	0.78	1.98	1.74	0.80
DHondtDO	**267.7**	**227.9**	0.76	**2.09**	1.79	0.77	**273.2**	**235.4**	0.77	**2.07**	1.80	0.78
EP-FuzzDA	249.6	222.9	**0.82**	1.99	1.59	0.69	253.3	229.9	**0.85**	1.96	1.60	0.70

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	252.9	182.6	0.51	1.89	**1.66**	**0.78**	255.2	187.5	0.54	1.88	**1.65**	**0.78**
AVG	241.1	203.3	0.73	1.89	1.52	0.69	244.7	207.9	0.74	1.88	1.54	0.71
DHondtDO	**253.7**	**211.7**	0.72	**1.97**	1.55	0.68	**258.2**	**217.2**	0.74	**1.96**	1.57	0.69
EP-FuzzDA	239.0	206.2	**0.76**	1.88	1.40	0.62	242.3	211.7	**0.78**	1.87	1.43	0.64

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	243.5	176.9	0.50	1.83	**1.53**	**0.72**	244.7	181.1	0.53	1.84	**1.54**	**0.73**
AVG	234.5	195.2	0.71	1.84	1.40	0.64	236.7	199.2	0.73	1.84	1.44	0.66
DHondtDO	**245.5**	**202.6**	0.71	**1.90**	1.42	0.62	**248.3**	**206.3**	0.72	**1.91**	1.46	0.64
EP-FuzzDA	232.4	197.4	**0.73**	1.83	1.30	0.57	234.5	201.2	**0.75**	1.83	1.35	0.60

Table 7.3: Results of offline **long-term** evaluation on **MovieLens25M** dataset. The best results are in bold, second-best are underscored.

Interestingly, the results for the long-term scenario are different from the weighted scenario. We would expect that if an algorithm can better correlate the provided relevance with the provided weights, then it will perform better across the board in the long-term scenario. But, as we can see in Table 7.3, DHondtDO dominates in mean and min AR metrics. EP-FuzzDA dominates in M/M AR metric. It balances the preference better than other algorithms, as discussed in the weighted scenario. This leads to better M/M AR performance because it provides fairer recommendations but for the price of decreased mean performance. We have checked the weights calculated after each round, and for EP-FuzzDA, they stay the most uniform, which tells us that the algorithm is doing a good job of maintaining fairness across the group members.

The results look analogical for the additional datasets, the KGRec dataset (Table A.9), the MovieLens1M dataset (Table A.10), and the Spotify dataset (Table A.12) except for the Netflix dataset (Table A.11), where for groups with sizes over 6, the fairness performance of the M/M AR metric drops slightly.

7.4 Discussion

The GFAR algorithm performs poorly in our tests compared to the original research in [45]. This is probably caused by the different evaluation types as mentioned in Subsection 7.1.1, we use the decoupled evaluation, contrary to the original paper where they used the coupled evaluation. As discussed in our research in [59], it is possible that due to GFAR's use of Borda count as the relevance selection criteria, there could have been a popularity bias at play. The coupled evaluation requires that we have data in the testing part of our dataset for our recommended items. We are likelier to have a rating for popular items due to the popularity bias. GFAR commonly selects items that are best or among the best items for a user. Therefore the probability that we have data in the testing data (for coupled) is higher. This can lead to an unfair advantage in the coupled evaluation compared to other algorithms. On the other hand, in decoupled evaluation, we have estimated ratings for all pairs of users and items. Therefore, we can equally evaluate the selected items for all algorithms without this popularity bias affecting our results.

Another explanation for why GFAR is not performing well in our experiments can be the parameter selection. The algorithm calculates the probability of user satisfaction after each recommended item, which is then used to adjust the relevance probability of the candidate items in the next step. But for a higher number of maximal relevance items (how many items from the top-p of each candidate will be awarded the Borda rating), these probabilities are small and therefore push very little to balance the inter-group relevance inequalities from the previously recommended items. If one person in a big group differs in their preference from the rest, then GFAR won't push hard enough to satisfy this person at some point, especially if other members are in preference consensus.

XPO and NPO algorithms are not performing well in our setup. Similar to the already discussed GFAR algorithm. For all experiments, we use a randomly selected subset of 1000 candidate items (for each user, then joining these candidate lists together) from which we select the top 10. This is an optimization for performance reasons. Unfortunately, XPO and NPO do not scale well in settings where many candidate items are presented to the algorithm. This is due to internally considering all tuples of the candidate items and comparing them using random trials (details can be found in Subsection 4.3.2). We had to therefore limit the candidate items to 30 (instead of 1000), and even that caused the XPO (and NPO) to remain the computationally slowest algorithm. An increase from 30 to only 100 candidate items resulted in 10 times worse computational performance, and we have therefore limited XPO and NPO algorithms to only 30 candidate items. Naturally, this leads to a lesser rating performance due to a lower selection pool.

The performance of our algorithm in the long-term evaluation was a little surprising. On the one hand, it can handle weighing well, as seen in the weighted scenario, but on the other hand, the average rating performance does not translate well to the long-term scenario. This is probably due to the algorithm pushing too hard to achieve the highest amount of fairness, even for a decrease in AR and nDCG mean and min performance. One possible way to suppress this could be to decrease the effect of the weights. However, then, again, the definition of

fairness must be considered. If we require the most balanced performance, then our algorithm performs the best (with regard to the AR M/M metric).

Additionally, the performance of our algorithm in the long-term evaluation could be explained by the way how the experiment was set up. We calculate the weights for the next run from the previously recommended items. Based on the ratio of average ratings between the group members, we set the weights for the next iteration. But we also exclude the already recommended items from the next run. These already recommended items are better than the rest of our candidates (why they have been selected). Therefore we, in a way, overestimate the potential future ratings that we can receive. Therefore it is possible that in the current setup of this scenario, the algorithms that do not conform to the desired weights that much will get an edge over our algorithm, which, as shown by the results of the weighted scenario, does balance the preference based on weights the most.

We have observed slightly better performance in the uniform scenario for the two implicit datasets (KGRec and Spotify) compared to the explicit datasets (Movie Lens and Netflix). This could be due to overall better performance on this type of data, or it could also be a difference in our underlying matrix factorization algorithm.

Lastly, the performance seems to be quite analogical (with small deviations) over the different datasets, which widely differ in size. The same cannot be said about XPO and NPO algorithms, which in our current setting (as mentioned above) perform considerably worse when we compare the performance on the KGRec dataset (smallest) and the Spotify dataset (largest).

7.5 Reproducibility

Throughout our research, we encountered multiple situations when the experiments from the literature were hard to replicate and reuse. We have therefore taken increased care to provide an easy and convenient way to run all the experiments found in this work.

Firstly, we have selected Python as our programming language due to being the most popular and most used language in the data science domain. Secondly, we have versioned all experiments in a single GitHub repository at github.com/LadislavMalecek/MasterThesisAnalysis. And Lastly, we have created a convenient reproducibility shell script `run_experiments.sh` which can be found in the Git repository, mentioned at the beginning of this chapter.

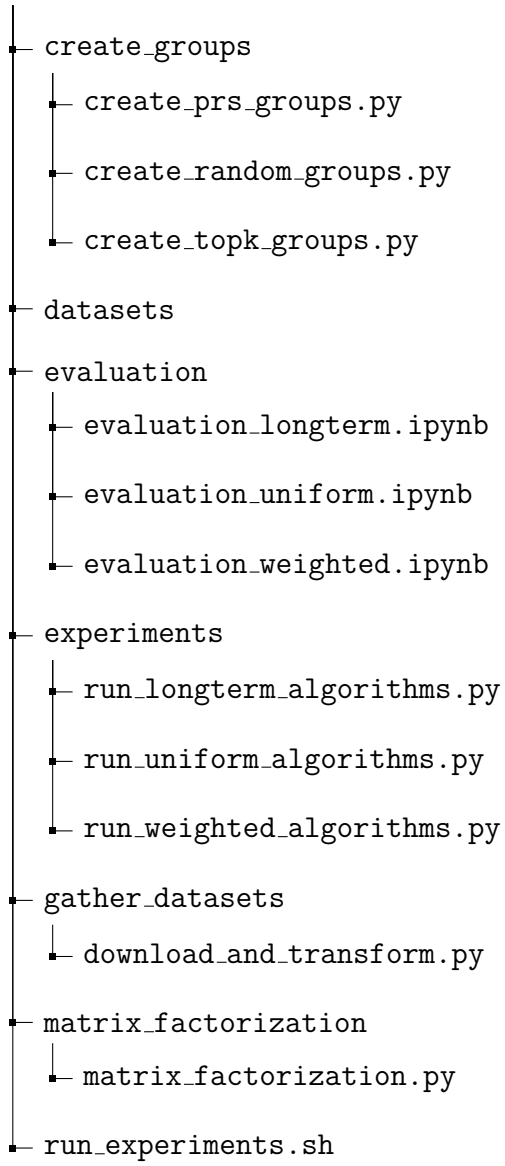
Requirements:

- Python 3.9
- Poetry - Python package manager
- 60 GB of disk space - storing the datasets and intermediate results
- 16 GB of ram (less if only running the smaller datasets (KGrec, Movielens-small)
- internet connection

After installing Python and poetry, clone the repository and run the provided shell script. Some of our more compute-heavy calculations, such as the matrix factorization, can utilize multi-core processing. Therefore, using a multi-core machine is advised (but not necessary).

Documentation of the repository structure can be found also in `./readme.md`. Each script is self-documenting, use the parameter `--help` to get more information about all usable parameters and default values.

The git repository has the following structure (we show only the most important files, which are mentioned in Section 7.2):



8. Conclusion

We have developed and implemented a novel group recommendation method, EP-FuzzDA, based on the D'Hondt mandate allocation algorithm. It is an aggregation method that works on top of single-user group recommendation systems.

To evaluate its performance, we have developed a fast and easy-to-use set of software components in Python, which simplifies reproducibility and future work with group recommendation systems. These tools provide a complete loop for decoupled evaluation. The separate components are a dataset downloader and processing tool, a synthetic group generation tool, a single-user recommendation system for generating the ground truths for decoupled evaluation, and a group recommendation tool into which a group aggregation recommender can be plugged into.

We have explored 8 group recommendation datasets from the related literature but determined that none are suitable enough to be used for our purposes of evaluating GRS. We have further described and analyzed 4 datasets without any group information but of suitable quality.

Further, we have designed a group generation method PRS (probability respecting similarity) that allows for a free scaling of the average similarity of the created artificial groups with the average inter-group similarity having a natural variance.

We have extensively evaluated our algorithm and seven other related algorithms on five datasets of various sizes, two group types, and three evaluation scenarios. It provides a good balance of fairness and item relevance, allows for direct use with group member importance weights, and provides the most correlated recommended item relevance with the desired weights. Additionally, we showed that it could be applied to a long-term recommendation scenario to maintain fairness across multiple recommendation sessions, but currently the other algorithms are giving us a better performance for mean relevance scores.

Compared to other related work, the evaluation has been performed not only on small datasets such as KGRec and MovieLens1M but also on huge and more organic datasets such as Spotify and Netflix. All tools in the evaluation software components have been optimized enough so that the evaluation of these huge datasets is still computationally feasible (in a matter of hours).

8.1 Future work

We have focused on the fairness aspect of designing a group recommendation system. The main problem we have encountered is the nonexistence of datasets with organic group recommendation information. The step of synthetic group creation can cause a bias towards some algorithms that will be hard to address differently than having an organic group recommendation dataset. This can be solved by an online evaluation using a user study focused on long-term fairness preservation in realistic group settings.

Additional work needs to be put into evaluating the effect of different underlying recommender systems on the decoupled evaluation we have used. We have selected a popular matrix factorization algorithm with parameters set to

similar values as in the related literature. However, we have not performed any comparison of the GRS algorithms with different underlying single-user RS.

Further, more software development effort can be put into expanding and improving the software evaluation tools that we have created. They are easily usable but still quite tightly tailored to our needs. A more general framework could be designed based on our current work.

Additionally, the evaluation of long-term and weighted evaluation scenarios can be improved by designing a better set of metrics that more directly measure fairness. This requires to better define what fairness is in these weighted and long-term settings.

More work can be put into our algorithm to make it fairness configurable so that the trade-off between performance and fairness can be adjusted as required. This would probably help in the long-term scenario when the groups are balanced enough, and the algorithm could push more toward the average rating performance.

Bibliography

[1] Wikipedia. Recommender system — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Recommender%20system&oldid=1020619015`, 2021. [Online; accessed 01-May-2021].

[2] Alexander Felfernig, Ludovico Boratto, Martin Stettinger, and Marko Tkalčič. *Group recommender systems: An introduction*. Springer, 2018.

[3] Bashir Rastegarpanah, Krishna P Gummadi, and Mark Crovella. Fighting fire with fire: Using antidote data to improve polarization and fairness of recommender systems. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 231–239, 2019.

[4] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[5] Alexander Felfernig and Robin Burke. Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th international conference on Electronic commerce*, pages 1–10, 2008.

[6] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201, 1995.

[7] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, 1995.

[8] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[9] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[10] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[11] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[12] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[13] Luis M de Campos, Juan M Fernández-Luna, Juan F Huete, and Miguel A Rueda-Morales. Managing uncertainty in group recommending processes. *User Modeling and User-Adapted Interaction*, 19(3):207–242, 2009.

[14] Judith Masthoff. Group recommender systems: Combining individual models. In *Recommender systems handbook*, pages 677–702. Springer, 2011.

[15] Lin Xiao, Zhang Min, Zhang Yongfeng, Gu Zhaoquan, Liu Yiqun, and Ma Shaoping. Fairness-aware group recommendation with pareto-efficiency. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 107–115, 2017.

[16] Mark O'connor, Dan Cosley, Joseph A Konstan, and John Riedl. Polylens: A recommender system for groups of users. In *ECSCW 2001*, pages 199–218. Springer, 2001.

[17] Anthony Jameson and Barry Smyth. Recommendation to groups. In *The adaptive web*, pages 596–627. Springer, 2007.

[18] Fairness noun - definition.

[19] Google. Google books ngram viewer 2012 corpa.

[20] family=McAuliffe given=Katherine, family=Blake given=Peter R., and family=Warneken given=Felix. Do kids have a fundamental sense of fairness?

[21] Sarah F Brosnan and Frans BM De Waal. Monkeys reject unequal pay. *Nature*, 425(6955):297–299, 2003.

[22] Sarah F Brosnan and Frans BM de Waal. Evolution of responses to (un) fairness. *Science*, 346(6207), 2014.

[23] Daniel John Zizzo and Andrew J Oswald. Are people willing to pay to reduce others' incomes? *Annales d'Economie et de Statistique*, pages 39–65, 2001.

[24] John Corbit, Katherine McAuliffe, Tara C Callaghan, Peter R Blake, and Felix Warneken. Children's collaboration induces fairness rather than generosity. *Cognition*, 168:344–356, 2017.

[25] Dana Pessach and Erez Shmueli. Algorithmic fairness. *arXiv preprint arXiv:2001.09784*, 2020.

[26] European Union Agency for Fundamental Rights, European Union. Agency for Fundamental Rights. Council of Europe, Europäische Union Agentur für Grundrechte, Europarat, Danmark. European Court of Human Rights, and Europäischer Gerichtshof für Menschenrechte. *Handbook on European Non-discrimination Law*. UTB, Stuttgart, Germany, 2018.

[27] Jeffrey Dastin. Amazon scraps secret AI recruiting tool that showed bias against women, 10 2018.

[28] Felix Richter. Women's Representation in Big Tech, 07 2021.

[29] Taylor Telford. Apple Card algorithm sparks gender bias allegations against Goldman Sachs, 11 2019.

[30] ProPublica. How We Analyzed the COMPAS Recidivism Algorithm, 02 2020.

[31] Solon Barocas, Moritz Hardt, and Arvind Narayanan. Fairness in machine learning. *Nips tutorial*, 1:2, 2017.

[32] David Blaszka. Dangerous Feedback Loops in ML - Towards Data Science, 12 2019.

[33] R Kelly Garrett. Politically motivated reinforcement seeking: Reframing the selective exposure debate. *Journal of communication*, 59(4):676–699, 2009.

[34] Matteo Cinelli, Gianmarco De Francisci Morales, Alessandro Galeazzi, Walter Quattrociocchi, and Michele Starnini. The echo chamber effect on social media. *Proceedings of the National Academy of Sciences*, 118(9), 2021.

[35] Pablo Barberá, John T Jost, Jonathan Nagler, Joshua A Tucker, and Richard Bonneau. Tweeting from left to right: Is online political communication more than an echo chamber? *Psychological science*, 26(10):1531–1542, 2015.

[36] GitHub Copilot regurgitates valid secrets — Hacker News, 07 2021.

[37] Dotan Nahum. 3 Weeks into the GitHub CoPilot secrets leak - What have we learned, 08 2021.

[38] Steve Lohr. Netflix Cancels Contest Over Privacy Concerns, 03 2010.

[39] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.

[40] Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. In *2007 IEEE 23rd international conference on data engineering workshop*, pages 801–810. IEEE, 2007.

[41] Sainyam Galhotra, Romila Pradhan, and Babak Salimi. Explaining black-box algorithms using probabilistic contrastive counterfactuals. In *Proceedings of the 2021 International Conference on Management of Data*, pages 577–590, 2021.

[42] Wikipedia contributors. Accuracy and precision, 05 2022.

[43] Judith Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. In *Personalized digital television*, pages 93–141. Springer, 2004.

[44] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. Paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338, 2016.

[45] Mesut Kaya, Derek Bridge, and Nava Tintarev. Ensuring fairness in group recommendations by rank-sensitive balancing of relevance. In *Fourteenth ACM Conference on Recommender Systems*, pages 101–110, 2020.

[46] Dimitris Sacharidis. Top-n group recommendations with fairness. In *Proceedings of the 34th ACM/SIGAPP symposium on applied computing*, pages 1663–1670, 2019.

[47] Lin Xiao, Zhang Min, Zhang Yongfeng, Gu Zhaoquan, Liu Yiqun, and Ma Shaoping. Fairness-aware group recommendation with pareto-efficiency. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 107–115, 2017.

[48] Wikipedia contributors. D'Hondt method, 06 2022.

[49] Ladislav Peska and Stepan Balkar. Fuzzy d'hondt's algorithm for on-line recommendations aggregation. In *2nd Workshop on Online Recommender Systems and User Modeling*, pages 2–11. PMLR, 2019.

[50] Sergio Oramas, Vito Claudio Ostuni, Tommaso Di Noia, Xavier Serra, and Eugenio Di Sciascio. Sound and music recommendation with knowledge graphs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):1–21, 2016.

[51] Gabriel Vigliensoni and Ichiro Fujinaga. Identifying time zones in a large dataset of music listening logs. In *Proceedings of the first international workshop on Social media retrieval and analysis*, pages 27–32, 2014.

[52] Da Cao, Xiangnan He, Lianhai Miao, Yahui An, Chao Yang, and Richang Hong. Attentive group recommendation. In *The 41st International ACM SIGIR conference on research & development in information retrieval*, pages 645–654, 2018.

[53] Aravind Sankar, Yanhong Wu, Yuhang Wu, Wei Zhang, Hao Yang, and Hari Sundaram. Groupim: A mutual information maximization framework for neural group recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1279–1288, 2020.

[54] Tuan-Anh Nguyen Pham, Xutao Li, Gao Cong, and Zhenjie Zhang. A general graph-based model for recommendation in event-based social networks. In *2015 IEEE 31st international conference on data engineering*, pages 567–578. IEEE, 2015.

[55] Lucas Vinh Tran, Tuan-Anh Nguyen Pham, Yi Tay, Yiding Liu, Gao Cong, and Xiaoli Li. Interact and decide: Medley of sub-attention networks for effective group recommendation. In *Proceedings of the 42nd International ACM SIGIR conference on research and development in information retrieval*, pages 255–264, 2019.

[56] Xingjie Liu, Qi He, Yuanyuan Tian, Wang-Chien Lee, John McPherson, and Jiawei Han. Event-based social networks: linking the online and offline social worlds. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1032–1040, 2012.

[57] Fethi Fkih. Similarity measures for collaborative filtering-based recommender systems: Review and experimental comparison. *Journal of King Saud University-Computer and Information Sciences*, 2021.

[58] Ladislav Malecek and Ladislav Peska. Fairness-preserving group recommendations with user weighting. In *Adjunct Proceedings of the 29th ACM Conference on User Modeling, Adaptation and Personalization*, pages 4–9, 2021.

[59] Ladislav Peska and Ladislav Malecek. Coupled or decoupled evaluation for group recommendation methods? In *Perspectives@ RecSys*, 2021.

[60] Judith Masthoff. Group recommender systems: Combining individual models. In *Recommender systems handbook*, pages 677–702. Springer, 2011.

List of abbreviations

RS Recommender System

GRS Group Recommender System

ML Machine Learning

TP True positive

FP False positive

TN True negative

FN False negative

List of Figures

List of Tables

A. Additional Results

In this apendix, we present additional results for four datasets in all three evaluation scenarios.
For the uniform evaluation scenario, we present the following results:

- KGRec dataset (Table A.1),
- MovieLens1M dataset (Table A.2),
- Netflix dataset (Table A.3),
- Spotify dataset (Table A.4).

For the weighted evaluation scenario, we present the following results:

- KGRec dataset (Table A.5),
- MovieLens1M dataset (Table A.6),
- Netflix dataset (Table A.7),
- Spotify dataset (Table A.8).

For the long-term evaluation scenario, we present the following results:

- KGRec dataset (Table A.9),
- MovieLens1M dataset (Table A.10),
- Netflix dataset (Table A.11),
- Spotify dataset (Table A.12).

	PRS(M=1), group size s=2						PRS(M=4), group size s=2					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**6.84**	5.15	0.61	**0.79**	0.66	0.74	**7.80**	6.52	0.72	0.88	0.81	0.86
FAI	6.41	5.30	0.71	0.74	*0.67*	*0.83*	*7.54*	6.44	0.74	0.84	0.78	0.86
LM	6.19	*5.54*	0.81	0.74	0.67	0.80	7.48	6.83	0.83	0.86	0.80	*0.87*
XPO	6.13	5.05	0.71	0.69	0.61	0.79	7.08	5.99	0.73	0.77	0.70	0.84
NPO	5.64	4.52	0.67	0.63	0.54	0.75	6.66	5.56	0.72	0.72	0.65	0.82
GFAR	3.03	2.49	0.72	0.35	0.30	0.78	3.47	2.88	0.73	0.38	0.34	0.81
DHondtDO	6.68	5.71	*0.74*	0.78	**0.74**	**0.88**	7.73	*6.82*	*0.78*	**0.88**	**0.84**	**0.92**
EP-FuzzDA	*6.44*	**6.15**	**0.91**	*0.78*	0.71	0.84	7.51	**7.13**	**0.90**	*0.87*	*0.81*	0.87

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**6.22**	3.48	0.40	**0.69**	0.44	0.49	**7.28**	5.41	0.59	**0.81**	0.66	0.71
FAI	5.63	3.95	*0.55*	0.63	*0.48*	0.62	6.84	5.27	0.62	0.75	0.63	0.71
LM	5.00	*4.00*	0.65	0.59	0.48	0.67	6.69	*5.71*	0.73	0.76	*0.66*	*0.77*
XPO	5.38	3.78	0.54	0.59	0.45	0.64	6.39	4.90	0.62	0.69	0.57	0.72
NPO	4.66	3.14	0.51	0.51	0.37	0.58	5.63	4.20	0.59	0.60	0.48	0.67
GFAR	2.84	1.87	0.50	0.32	0.23	0.56	3.26	2.45	0.61	0.36	0.29	0.67
DHondtDO	5.93	4.12	0.53	0.68	0.52	*0.65*	7.18	5.76	*0.66*	0.81	0.70	0.77
EP-FuzzDA	*5.72*	**4.95**	**0.77**	*0.67*	**0.58**	**0.77**	*7.00*	**6.19**	**0.80**	*0.80*	**0.71**	**0.81**

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**5.78**	2.47	0.28	**0.64**	0.30	0.34	**7.03**	4.44	0.47	**0.77**	0.54	0.58
FAI	5.03	*3.01*	0.44	0.56	0.36	0.49	6.55	4.55	0.53	0.71	0.53	0.61
LM	4.19	3.01	0.54	0.49	*0.36*	0.56	6.09	*4.82*	0.64	0.68	*0.56*	0.68
XPO	4.75	2.91	*0.44*	0.52	0.34	*0.51*	5.96	4.17	0.53	0.63	0.48	0.62
NPO	3.93	2.26	0.40	0.43	0.27	0.46	5.06	3.40	0.49	0.53	0.39	0.57
GFAR	2.67	1.41	0.37	0.30	0.17	0.42	3.29	2.23	0.51	0.35	0.25	0.58
DHondtDO	5.42	3.09	0.41	*0.61*	0.39	0.49	6.89	4.96	*0.56*	0.76	0.60	*0.67*
EP-FuzzDA	*5.26*	**3.98**	**0.64**	0.61	**0.47**	**0.64**	*6.74*	**5.42**	**0.68**	*0.75*	**0.64**	**0.73**

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**5.20**	1.42	0.17	**0.58**	0.17	0.20	**6.70**	3.31	0.34	**0.73**	0.39	0.42
FAI	4.36	2.11	*0.32*	0.49	0.25	0.36	6.06	3.54	0.41	0.65	0.41	0.48
LM	3.35	*1.98*	0.40	0.39	*0.24*	0.43	5.38	*3.73*	0.51	0.60	*0.43*	0.55
XPO	4.00	1.94	0.32	0.44	0.23	*0.37*	5.35	3.26	*0.43*	0.56	0.37	0.50
NPO	3.14	1.39	0.27	0.34	0.16	0.31	4.31	2.42	0.37	0.45	0.27	0.44
GFAR	2.55	0.98	0.25	0.29	0.12	0.27	3.29	1.88	0.40	0.35	0.21	0.44
DHondtDO	4.72	1.76	0.23	*0.53*	0.23	0.29	6.49	3.80	0.42	*0.71*	0.46	*0.51*
EP-FuzzDA	*4.68*	**2.69**	**0.44**	0.54	**0.33**	**0.47**	*6.40*	**4.22**	**0.51**	0.71	**0.50**	**0.57**

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**4.92**	0.87	0.10	**0.54**	0.11	0.12	**6.40**	2.62	0.27	**0.69**	0.30	0.33
FAI	4.09	1.56	0.24	0.45	0.18	0.26	5.75	2.97	0.35	0.61	0.34	0.40
LM	2.88	*1.47*	**0.32**	0.34	*0.18*	**0.35**	4.79	*3.00*	**0.43**	0.53	*0.35*	**0.47**
XPO	3.62	1.43	*0.24*	0.39	0.17	*0.28*	4.85	2.68	*0.37*	0.51	0.30	*0.42*
NPO	2.78	0.95	0.19	0.30	0.11	0.22	3.79	1.90	0.31	0.39	0.21	0.35
GFAR	2.52	0.72	0.17	0.28	0.09	0.19	3.25	1.59	0.32	0.35	0.18	0.36
DHondtDO	*4.41*	1.09	0.14	*0.49*	0.15	0.19	6.13	3.02	0.33	*0.67*	0.36	0.40
EP-FuzzDA	4.42	**1.87**	0.30	0.51	**0.24**	0.34	*6.09*	**3.46**	0.41	0.67	**0.40**	0.46

Table A.1: Results of offline **uniform** evaluation on **KGRec** dataset. The best results are in bold, the second-best are underscored, and the third-best results are in italic.

PRS(M=1), group size s=2

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**54.40**	47.21	0.78	0.89	0.85	0.91
FAI	50.37	46.36	0.86	0.83	0.79	0.91
LM	51.69	50.25	0.95	0.86	0.80	0.87
XPO	50.38	46.22	0.85	0.82	0.78	0.90
NPO	49.08	44.97	0.85	0.80	0.75	0.88
GFAR	46.21	43.07	*0.88*	0.77	0.74	*0.91*
DHondtDO	54.27	*48.58*	0.82	**0.89**	**0.87**	**0.95**
EP-FuzzDA	*52.70*	**51.37**	**0.95**	*0.88*	*0.83*	0.89

PRS(M=4), group size s=2

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**54.90**	48.60	0.81	0.89	0.85	0.92
FAI	50.96	47.33	0.87	0.83	0.79	*0.92*
LM	52.40	51.09	0.95	0.86	0.81	0.89
XPO	51.03	47.18	0.87	0.82	0.78	0.91
NPO	49.66	45.70	0.86	0.80	0.75	0.88
GFAR	46.71	43.87	*0.89*	0.77	0.74	0.92
DHondtDO	54.79	*49.86*	0.85	**0.89**	**0.87**	**0.95**
EP-FuzzDA	*53.50*	**52.44**	**0.96**	*0.88*	*0.83*	0.90

PRS(M=1), group size s=3

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**52.06**	43.67	0.72	0.84	0.79	0.89
FAI	46.83	41.90	0.81	0.76	0.68	0.82
LM	49.12	46.83	**0.91**	0.81	0.72	0.81
XPO	47.62	42.26	0.80	0.77	0.69	0.82
NPO	45.69	40.28	0.79	0.74	0.65	0.79
GFAR	45.74	41.17	*0.82*	0.74	0.67	0.82
DHondtDO	51.92	*45.09*	0.77	**0.84**	**0.81**	**0.92**
EP-FuzzDA	*50.68*	**47.83**	0.90	*0.83*	*0.76*	*0.84*

PRS(M=4), group size s=3

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**52.08**	44.81	0.75	0.85	0.80	0.89
FAI	47.05	42.46	0.82	0.77	0.69	0.83
LM	49.53	47.50	0.92	0.82	0.74	0.84
XPO	47.80	42.75	0.81	0.78	0.70	0.83
NPO	45.91	40.70	0.80	0.75	0.66	0.81
GFAR	45.98	41.60	*0.83*	0.75	0.68	0.84
DHondtDO	51.97	*46.09*	0.80	**0.85**	**0.81**	**0.92**
EP-FuzzDA	*50.90*	**48.70**	**0.92**	*0.84*	*0.77*	*0.86*

PRS(M=1), group size s=4

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**50.55**	41.73	0.69	0.82	0.75	0.86
FAI	44.86	39.11	0.77	0.73	0.62	0.76
LM	47.75	44.72	**0.88**	0.78	0.68	0.78
XPO	46.11	39.88	0.76	0.74	0.64	0.76
NPO	43.82	37.64	0.75	0.71	0.60	0.74
GFAR	44.38	39.06	*0.79*	0.72	0.62	0.77
DHondtDO	50.44	*43.08*	0.74	**0.82**	**0.76**	**0.88**
EP-FuzzDA	*49.40*	**45.48**	0.86	*0.81*	*0.72*	*0.81*

PRS(M=4), group size s=4

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**50.53**	42.84	0.72	0.82	0.76	0.87
FAI	45.06	39.65	0.78	0.73	0.63	0.77
LM	47.96	45.31	**0.89**	0.79	0.70	0.80
XPO	46.26	40.32	0.77	0.75	0.65	0.78
NPO	44.11	38.00	0.76	0.72	0.61	0.75
GFAR	44.47	39.48	*0.80*	0.73	0.64	0.79
DHondtDO	50.44	*44.02*	0.77	**0.82**	**0.77**	**0.89**
EP-FuzzDA	*49.52*	**46.33**	0.88	*0.81*	*0.73*	*0.82*

PRS(M=1), group size s=6

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**48.65**	39.90	0.68	0.79	0.70	0.81
FAI	42.74	36.15	0.73	0.69	0.55	0.69
LM	46.07	42.23	**0.84**	0.75	0.64	0.74
XPO	44.26	37.00	0.72	0.72	0.57	0.69
NPO	41.96	34.98	0.71	0.68	0.54	0.68
GFAR	43.58	37.22	*0.75*	0.71	0.58	0.70
DHondtDO	48.59	*40.91*	0.72	**0.79**	**0.71**	**0.82**
EP-FuzzDA	*47.84*	**42.73**	0.82	*0.78*	*0.67*	*0.77*

PRS(M=4), group size s=6

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**49.03**	41.11	0.70	0.79	0.70	0.81
FAI	43.15	36.61	0.74	0.69	0.56	0.70
LM	46.59	43.13	**0.85**	0.76	0.64	0.75
XPO	44.55	37.40	0.72	0.71	0.57	0.70
NPO	42.14	35.10	0.71	0.68	0.54	0.67
GFAR	43.83	37.76	*0.76*	0.71	0.58	0.71
DHondtDO	48.96	*42.10*	0.74	**0.79**	**0.70**	**0.82**
EP-FuzzDA	*48.29*	**43.92**	0.84	*0.78*	*0.67*	*0.77*

PRS(M=1), group size s=8

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**47.55**	38.28	0.66	0.77	0.66	0.77
FAI	41.61	34.35	0.70	0.68	0.51	0.64
LM	45.09	40.51	**0.81**	0.74	0.60	0.71
XPO	42.87	35.03	0.69	0.70	0.53	0.64
NPO	41.02	33.28	0.68	0.67	0.50	0.62
GFAR	43.34	35.96	*0.72*	0.71	0.55	0.66
DHondtDO	47.50	*39.21*	0.70	**0.77**	**0.67**	**0.77**
EP-FuzzDA	*46.87*	**40.67**	0.78	*0.77*	*0.64*	*0.73*

PRS(M=4), group size s=8

	AR			nDCG		
	mean	min	M/M	mean	min	M/M
AVG	**48.06**	39.78	0.69	0.77	0.66	0.77
FAI	41.94	34.86	0.71	0.67	0.52	0.65
LM	45.70	41.57	**0.82**	0.74	0.61	0.73
XPO	43.22	35.26	0.69	0.69	0.53	0.65
NPO	41.22	33.69	0.69	0.66	0.50	0.64
GFAR	43.60	36.58	*0.73*	0.70	0.55	0.67
DHondtDO	47.99	*40.65*	0.72	**0.77**	**0.67**	**0.78**
EP-FuzzDA	*47.44*	**42.14**	0.80	*0.76*	*0.64*	*0.74*

Table A.2: Results of offline **uniform** evaluation on **MovieLens1M** dataset. The best results are in bold, the second-best are underscored, and the third-best results are in italic.

	PRS(M=1), group size s=2						PRS(M=4), group size s=2					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**84.16**	58.26	0.59	0.82	0.73	0.80	**84.29**	59.08	0.60	0.82	0.73	0.81
FAI	69.96	58.13	0.74	0.71	0.65	*0.84*	70.28	58.60	0.74	0.71	0.65	0.84
LM	73.70	68.77	0.88	0.77	0.67	0.77	74.14	69.36	0.88	0.78	0.68	0.78
XPO	69.28	57.15	0.74	0.70	0.63	0.82	69.89	57.75	0.73	0.70	0.63	0.82
NPO	67.07	54.49	0.72	0.68	0.59	0.78	67.23	55.16	0.73	0.68	0.60	0.80
GFAR	57.45	49.38	*0.78*	0.62	0.56	0.84	57.85	50.16	*0.78*	0.62	0.57	*0.84*
DHondtDO	82.84	*64.43*	0.68	**0.83**	**0.79**	**0.92**	82.93	*65.02*	0.69	**0.83**	**0.79**	**0.93**
EP-FuzzDA	*75.21*	**71.09**	**0.90**	*0.80*	*0.71*	0.80	*75.61*	**71.71**	**0.90**	*0.80*	*0.71*	0.80

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**77.52**	46.78	0.45	0.74	0.63	*0.73*	**77.42**	48.42	0.47	0.73	0.63	*0.75*
FAI	59.33	46.29	0.65	0.61	0.49	0.68	59.24	47.09	0.67	0.60	0.48	0.67
LM	65.16	57.81	**0.79**	0.68	0.55	0.67	65.80	58.90	**0.81**	0.68	0.55	0.69
XPO	60.04	45.76	0.63	0.60	0.49	0.69	59.85	46.33	0.64	0.60	0.49	0.69
NPO	56.52	42.17	0.61	0.58	0.44	0.63	56.86	42.29	0.61	0.58	0.45	0.64
GFAR	53.94	42.97	*0.67*	0.58	0.46	0.69	54.43	43.81	*0.69*	0.58	0.47	0.70
DHondtDO	75.56	*52.75*	0.55	**0.74**	**0.70**	**0.88**	75.60	*53.84*	0.57	**0.74**	**0.69**	**0.88**
EP-FuzzDA	*69.98*	**58.82**	0.76	*0.73*	*0.61*	0.75	*70.45*	**60.54**	0.78	*0.73*	*0.62*	0.76

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**71.38**	41.70	0.40	0.69	0.58	*0.71*	**73.07**	44.19	0.42	0.68	0.57	*0.71*
FAI	53.04	39.81	0.60	0.55	0.40	0.57	54.11	41.04	0.61	0.54	0.39	0.58
LM	59.48	**51.32**	**0.75**	0.63	0.48	0.61	61.49	53.56	**0.76**	0.63	0.49	0.64
XPO	54.12	39.80	0.58	0.56	0.40	0.58	55.44	40.99	0.59	0.55	0.40	0.58
NPO	50.67	35.99	0.55	0.53	0.36	0.53	51.75	36.81	0.55	0.52	0.36	0.54
GFAR	52.71	39.76	*0.61*	0.56	0.41	0.59	53.45	41.12	*0.62*	0.55	0.41	0.60
DHondtDO	69.44	*46.17*	0.49	**0.70**	**0.63**	**0.84**	71.29	*48.99*	0.52	**0.69**	**0.63**	**0.83**
EP-FuzzDA	*65.40*	51.26	0.67	*0.69*	*0.56*	0.72	*67.24*	**54.59**	0.71	*0.68*	*0.56*	0.72

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**66.12**	37.91	0.35	*0.63*	0.50	0.65	**66.42**	39.19	0.37	0.63	0.50	*0.66*
FAI	47.56	33.01	*0.51*	0.50	0.29	0.44	47.67	33.11	*0.52*	0.49	0.29	0.44
LM	54.19	**45.56**	**0.70**	0.57	0.40	0.53	54.99	46.56	**0.71**	0.57	0.41	0.56
XPO	49.58	33.23	0.49	0.51	0.31	0.44	49.92	32.90	0.49	0.51	0.30	0.44
NPO	45.89	29.67	0.47	0.48	0.27	0.40	45.77	29.52	0.47	0.47	0.27	0.41
GFAR	49.81	33.83	0.50	0.52	0.31	0.45	49.76	34.11	0.52	0.52	0.31	0.46
DHondtDO	64.32	*40.75*	0.44	**0.64**	**0.54**	**0.74**	64.42	*41.45*	0.46	**0.63**	**0.53**	**0.73**
EP-FuzzDA	*61.22*	44.83	0.58	0.63	*0.49*	*0.64*	*61.68*	**46.78**	0.62	*0.63*	*0.49*	0.66

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**61.92**	35.06	0.32	*0.60*	0.45	0.61	**62.78**	*36.52*	0.35	0.60	0.45	0.61
FAI	44.06	28.14	0.45	0.47	0.23	0.34	44.56	28.79	0.47	0.46	0.24	0.36
LM	50.22	**41.72**	**0.68**	0.54	0.35	0.47	51.27	**43.11**	**0.69**	0.53	0.36	0.50
XPO	46.46	28.31	0.42	0.49	0.24	0.34	47.02	28.17	0.42	0.48	0.25	0.36
NPO	42.32	25.09	0.41	0.45	0.21	0.31	42.93	25.16	0.41	0.44	0.21	0.32
GFAR	46.09	29.94	*0.47*	0.49	0.26	0.37	46.62	30.54	*0.48*	0.48	0.26	0.38
DHondtDO	59.84	*35.57*	0.39	0.60	**0.47**	**0.64**	60.52	36.01	0.41	*0.59*	**0.46**	**0.64**
EP-FuzzDA	*57.66*	40.88	0.54	**0.60**	*0.44*	*0.58*	*58.82*	42.91	0.57	**0.60**	*0.44*	*0.61*

Table A.3: Results of offline **uniform** evaluation on **Netflix** dataset. The best results are in bold, the second-best are underscored, and the third-best results are in italic.

	PRS(M=1), group size s=2						PRS(M=4), group size s=2					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**4.13**	2.05	0.33	*0.65*	0.38	0.44	**4.81**	3.00	0.43	*0.77*	0.60	0.65
FAI	3.75	2.68	*0.51*	**0.68**	**0.61**	**0.81**	4.70	3.45	*0.54*	**0.80**	**0.73**	**0.84**
LM	2.41	2.00	0.64	0.44	0.33	0.56	3.89	3.20	0.66	0.68	0.57	0.69
XPO	3.46	*2.37*	0.49	0.57	*0.49*	*0.75*	4.20	2.99	0.52	0.65	0.58	0.79
NPO	3.00	1.98	0.47	0.50	0.40	0.67	3.70	2.59	0.51	0.58	0.50	0.74
GFAR	1.02	0.66	0.46	0.16	0.13	0.66	1.43	0.97	0.51	0.22	0.18	0.71
DHondtDO	*3.59*	2.24	0.38	0.59	0.44	0.57	*4.52*	*3.23*	0.49	0.74	*0.66*	0.76
EP-FuzzDA	3.29	**2.90**	**0.70**	0.66	0.57	0.75	4.17	**3.64**	**0.71**	0.77	0.68	*0.79*

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**3.53**	1.04	0.15	*0.50*	0.17	0.20	**4.46**	2.26	0.30	*0.65*	0.42	0.48
FAI	3.02	1.68	*0.33*	**0.53**	0.39	0.57	4.22	2.74	*0.42*	**0.67**	0.52	0.64
LM	1.53	1.03	0.38	0.27	0.17	0.38	3.11	2.26	0.48	0.51	0.38	0.53
XPO	2.81	*1.44*	0.30	0.45	*0.32*	*0.56*	3.73	2.24	0.38	0.55	0.42	*0.63*
NPO	2.20	1.05	0.27	0.35	0.22	0.45	3.00	1.70	0.34	0.44	0.31	0.53
GFAR	0.93	0.35	0.22	0.15	0.07	0.37	1.32	0.68	0.31	0.19	0.12	0.47
DHondtDO	*2.83*	1.11	0.17	0.45	0.20	0.28	*4.02*	*2.38*	0.33	0.61	*0.44*	0.53
EP-FuzzDA	2.70	**1.93**	**0.50**	0.52	**0.41**	**0.65**	3.88	**2.93**	**0.55**	0.66	**0.54**	**0.69**

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**3.14**	0.54	0.07	*0.43*	0.08	0.10	**4.02**	1.66	0.21	*0.57*	0.30	0.34
FAI	2.57	1.05	*0.22*	**0.43**	0.26	0.42	3.72	2.03	*0.31*	**0.58**	0.39	*0.50*
LM	1.18	0.60	0.22	0.20	0.10	0.27	2.58	1.62	0.35	0.41	0.27	0.42
XPO	*2.41*	*0.88*	0.19	0.38	*0.21*	*0.40*	3.30	1.66	0.28	0.48	*0.32*	0.50
NPO	1.77	0.58	0.15	0.28	0.12	0.29	2.51	1.17	0.24	0.36	0.22	0.40
GFAR	0.89	0.22	0.11	0.13	0.04	0.21	1.29	0.54	0.21	0.19	0.10	0.33
DHondtDO	2.38	0.52	0.06	0.37	0.09	0.12	*3.54*	*1.68*	0.22	0.53	0.31	0.36
EP-FuzzDA	2.38	**1.27**	**0.34**	0.43	**0.27**	**0.46**	3.50	**2.25**	**0.42**	0.58	**0.42**	**0.55**

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**2.62**	0.17	0.02	**0.34**	0.02	0.02	**3.58**	0.98	0.12	**0.49**	0.17	0.19
FAI	2.00	0.48	0.11	*0.33*	**0.13**	**0.25**	3.20	1.25	*0.19*	*0.48*	0.23	0.31
LM	0.84	0.25	*0.10*	0.14	0.05	0.17	1.99	0.97	0.20	0.30	0.16	0.29
XPO	1.89	*0.36*	0.08	0.29	*0.10*	*0.19*	2.81	*1.03*	0.17	0.39	*0.19*	*0.30*
NPO	1.26	0.21	0.06	0.20	0.04	0.11	1.92	0.62	0.14	0.27	0.11	0.23
GFAR	0.80	0.07	0.03	0.12	0.01	0.05	1.22	0.34	0.10	0.17	0.05	0.17
DHondtDO	1.85	0.05	-0.03	0.28	-0.02	-0.07	3.04	0.94	0.10	0.45	0.16	0.16
EP-FuzzDA	*1.98*	**0.54**	**0.15**	0.34	0.11	0.21	*3.15*	**1.49**	**0.26**	0.49	**0.27**	**0.37**

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG	**2.21**	0.02	0.00	**0.29**	-0.02	-0.02	**3.30**	*0.69*	0.08	**0.44**	0.10	0.11
FAI	*1.63*	**0.27**	0.07	*0.27*	**0.09**	**0.17**	*2.83*	0.90	0.14	*0.41*	0.16	0.23
LM	0.53	0.09	*0.05*	0.09	0.02	0.11	1.63	0.63	*0.13*	0.23	0.11	*0.21*
XPO	1.51	*0.13*	0.03	0.23	*0.04*	0.08	2.43	0.65	0.10	0.32	*0.11*	0.18
NPO	0.96	0.05	0.01	0.15	0.01	0.02	1.63	0.38	0.08	0.22	0.06	0.12
GFAR	0.68	0.00	-0.01	0.10	-0.00	-0.02	1.22	0.24	0.06	0.16	0.03	0.08
DHondtDO	1.48	-0.13	-0.06	0.23	-0.05	-0.13	2.71	0.53	0.04	0.38	0.08	0.06
EP-FuzzDA	1.64	0.23	**0.07**	0.28	0.04	*0.09*	2.88	**1.04**	**0.17**	0.43	**0.18**	**0.25**

Table A.4: Results of offline **uniform** evaluation on **Spotify** dataset. The best results are in bold, the second-best are underscored, and the third-best results are in italic.

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**6.22**	0.01	0.19	**0.69**	0.01	0.18	**7.28**	0.01	0.17	**0.81**	0.01	0.16
AVG	5.91	0.45	0.15	0.66	0.45	0.13	7.05	0.35	0.14	0.79	0.35	0.12
DHondtDO	5.75	0.45	0.13	0.66	0.45	0.12	7.01	0.35	0.13	0.79	0.35	0.13
EP-FuzzDA	5.65	**0.64**	**0.09**	0.66	**0.64**	**0.09**	6.82	**0.53**	**0.11**	0.78	**0.53**	**0.11**

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**5.78**	-0.01	0.15	**0.64**	-0.01	0.14	**7.03**	-0.00	0.13	**0.77**	-0.00	0.13
AVG	5.48	0.39	0.12	0.61	0.39	0.11	6.83	0.30	0.11	0.75	0.30	0.10
DHondtDO	5.24	0.40	0.11	0.59	0.40	0.10	6.74	0.30	0.11	0.74	0.30	0.10
EP-FuzzDA	5.23	**0.53**	**0.08**	0.60	**0.53**	**0.08**	6.64	**0.42**	**0.09**	0.74	**0.42**	**0.09**

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**5.20**	-0.01	0.11	**0.58**	-0.01	0.11	**6.70**	-0.03	0.10	**0.73**	-0.03	0.09
AVG	4.94	0.32	0.09	0.55	0.32	0.09	6.52	0.21	0.08	0.71	0.21	0.08
DHondtDO	4.60	0.32	0.08	0.52	0.32	0.08	6.38	0.21	0.08	0.69	0.21	0.08
EP-FuzzDA	4.69	**0.43**	**0.06**	0.54	**0.43**	**0.07**	6.35	**0.28**	**0.07**	0.70	**0.28**	**0.07**

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**4.92**	0.01	0.08	**0.54**	0.01	0.08	**6.40**	0.01	0.07	**0.69**	0.01	0.07
AVG	4.73	0.25	0.07	0.52	0.25	0.07	6.26	0.19	0.07	0.68	0.19	0.06
DHondtDO	4.32	0.25	0.07	0.48	0.25	0.06	6.06	0.20	0.06	0.66	0.20	0.06
EP-FuzzDA	4.49	**0.32**	**0.05**	0.51	**0.32**	**0.06**	6.10	**0.25**	**0.06**	0.67	**0.25**	**0.06**

Table A.5: Results of offline **weighted** evaluation on **KGRec** dataset. The best results are in bold. The second-best are underscored.

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**52.06**	-0.02	0.15	**0.84**	-0.02	0.14	**52.08**	-0.00	0.15	**0.85**	-0.00	0.14
AVG	50.62	0.47	0.12	0.82	0.47	0.11	50.73	0.52	0.12	0.83	0.52	0.11
DHondtDO	50.73	0.48	0.12	0.83	0.48	0.12	50.92	0.53	0.12	0.83	0.53	0.12
EP-FuzzDA	48.97	**0.78**	**0.10**	0.81	**0.78**	**0.10**	49.24	**0.79**	**0.10**	0.81	**0.79**	**0.10**

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**50.55**	0.02	0.12	**0.82**	0.02	0.11	**50.53**	-0.01	0.12	**0.82**	-0.01	0.11
AVG	49.24	0.50	0.10	0.80	0.50	0.09	49.31	0.50	0.10	0.80	0.50	0.09
DHondtDO	49.42	0.51	0.10	0.80	0.51	0.09	49.43	0.51	0.10	0.81	0.51	0.09
EP-FuzzDA	48.01	**0.69**	**0.08**	0.78	**0.69**	**0.09**	48.09	**0.70**	**0.08**	0.79	**0.70**	**0.09**

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**48.65**	-0.01	0.08	**0.79**	-0.01	0.08	**49.03**	0.02	0.08	**0.79**	0.02	0.08
AVG	47.57	0.46	0.07	0.77	0.46	0.07	47.92	0.50	0.07	0.77	0.50	0.07
DHondtDO	47.71	0.47	0.07	0.77	0.47	0.07	48.09	0.51	0.07	0.77	0.51	0.07
EP-FuzzDA	46.57	**0.60**	**0.06**	0.76	**0.60**	**0.07**	46.99	**0.62**	**0.06**	0.76	**0.62**	**0.07**

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**47.55**	0.00	0.06	**0.77**	0.00	0.06	**48.06**	0.01	0.06	**0.77**	0.01	0.06
AVG	46.61	0.43	0.05	0.76	0.43	0.05	47.13	0.46	0.05	0.75	0.46	0.05
DHondtDO	46.72	0.44	0.05	0.76	0.44	0.05	47.23	0.47	0.05	0.75	0.47	0.05
EP-FuzzDA	45.79	**0.54**	**0.05**	0.75	**0.54**	**0.05**	46.31	**0.57**	**0.05**	0.74	**0.57**	**0.05**

Table A.6: Results of offline **weighted** evaluation on **MovieLens1M** dataset. The best results are in bold. The second-best are underscored.

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
		AR			nDCG			AR			nDCG	
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**77.52**	0.02	0.18	**0.74**	0.02	0.15	**77.42**	0.04	0.18	**0.73**	0.04	0.15
AVG	73.52	0.41	0.14	0.71	0.41	0.10	73.42	0.47	0.13	0.70	0.47	0.09
DHondtDO	72.34	0.41	0.13	0.71	0.41	0.10	72.75	0.48	0.12	0.71	0.48	0.10
EP-FuzzDA	68.32	**0.67**	**0.09**	0.70	**0.67**	**0.09**	68.47	**0.71**	**0.08**	0.69	**0.71**	**0.09**

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
		AR			nDCG			AR			nDCG	
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**71.38**	0.01	0.14	**0.69**	0.01	0.12	**73.07**	0.00	0.14	**0.68**	0.00	0.12
AVG	67.50	0.42	0.11	0.66	0.42	0.08	68.69	0.46	0.11	0.65	0.46	0.08
DHondtDO	66.35	0.44	0.10	0.67	0.44	0.08	67.62	0.49	0.10	0.66	0.49	0.08
EP-FuzzDA	63.59	**0.62**	**0.08**	0.66	**0.62**	**0.08**	65.49	**0.64**	**0.07**	0.65	**0.64**	**0.08**

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
		AR			nDCG			AR			nDCG	
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**66.12**	0.02	0.10	**0.63**	0.02	0.08	**66.42**	0.00	0.09	**0.63**	0.00	0.08
AVG	62.65	0.41	0.08	0.61	0.41	**0.06**	62.84	0.43	0.07	0.60	0.43	0.06
DHondtDO	61.41	0.44	0.07	0.61	0.44	0.06	61.43	0.46	0.07	0.61	0.46	0.06
EP-FuzzDA	59.92	**0.55**	**0.06**	0.61	**0.55**	0.06	60.24	**0.57**	**0.05**	0.61	**0.57**	**0.06**

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
		AR			nDCG			AR			nDCG	
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**61.92**	0.02	0.07	**0.60**	0.02	0.06	**62.78**	0.01	0.07	**0.60**	0.01	0.06
AVG	58.78	0.41	0.06	0.58	0.41	**0.05**	59.40	0.44	0.06	0.57	0.44	**0.04**
DHondtDO	57.16	0.44	0.06	0.58	0.44	0.05	57.86	0.46	0.05	0.57	0.46	0.05
EP-FuzzDA	56.54	**0.53**	**0.04**	0.58	**0.53**	0.05	57.52	**0.55**	**0.04**	0.58	**0.55**	0.05

Table A.7: Results of offline **weighted** evaluation on **Netflix** dataset. The best results are in bold. The second-best are underscored.

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
		AR			nDCG			AR			nDCG	
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**3.53**	0.02	0.28	**0.50**	0.02	0.26	**4.46**	-0.00	0.24	**0.65**	-0.00	0.21
AVG	3.28	0.38	0.24	0.49	0.38	0.21	4.25	0.30	0.21	0.63	0.30	0.17
DHondtDO	2.73	0.38	0.24	0.44	0.38	0.26	3.91	0.30	0.20	0.60	0.30	0.23
EP-Fuzz-DA	2.69	**0.57**	**0.12**	0.50	**0.57**	**0.09**	3.81	**0.46**	**0.13**	0.64	**0.46**	**0.11**

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
		AR			nDCG			AR			nDCG	
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**3.14**	0.00	0.23	**0.43**	0.00	0.22	**4.02**	-0.00	0.20	**0.57**	-0.00	0.17
AVG	2.89	0.37	0.21	0.40	0.37	0.19	3.81	0.31	0.17	0.55	0.31	0.14
DHondtDO	2.31	0.32	0.20	0.36	0.32	0.21	3.41	0.30	0.17	0.52	0.30	0.17
EP-Fuzz-DA	2.40	**0.49**	**0.10**	0.42	**0.49**	**0.08**	3.47	**0.43**	**0.10**	0.56	**0.43**	**0.09**

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
		AR			nDCG			AR			nDCG	
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**2.62**	0.03	0.17	**0.34**	0.03	0.16	**3.58**	-0.00	0.14	**0.49**	-0.00	0.12
AVG	2.42	0.37	0.15	0.32	0.37	0.14	3.39	0.31	0.12	0.47	0.31	0.11
DHondtDO	1.82	0.30	0.15	0.28	0.30	0.15	2.99	0.26	0.12	0.43	0.26	0.12
EP-Fuzz-DA	2.01	**0.44**	**0.08**	0.33	**0.44**	**0.07**	3.13	**0.36**	**0.08**	0.48	**0.36**	**0.07**

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
		AR			nDCG			AR			nDCG	
	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE	mean	corr	MAE
AVG-U	**2.21**	-0.02	0.13	**0.29**	-0.02	0.13	**3.30**	0.02	0.11	**0.44**	0.02	0.10
AVG	2.03	0.32	0.12	0.27	0.32	0.11	3.16	0.28	0.10	0.42	0.28	0.09
DHondtDO	1.44	0.23	0.11	0.22	0.23	0.11	2.65	0.23	0.09	0.37	0.23	0.09
EP-Fuzz-DA	1.67	**0.38**	**0.06**	0.27	**0.38**	**0.06**	2.89	**0.33**	**0.06**	0.42	**0.33**	**0.06**

Table A.8: Results of offline **weighted** evaluation on **Spotify** dataset. The best results are in bold. The second-best are underscored.

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	22.00	13.45	0.44	1.48	1.07	0.58	25.69	18.96	0.58	1.73	1.47	0.74
AVG	20.60	17.36	0.73	1.46	1.30	**0.80**	24.84	21.27	0.75	1.72	1.56	**0.83**
DHondtDO	**28.33**	**24.47**	0.76	**1.84**	**1.62**	0.78	**35.16**	**30.71**	0.77	**2.21**	**2.00**	0.82
EP-FuzzDA	20.09	18.13	**0.84**	1.44	1.26	0.77	24.38	21.92	**0.83**	1.70	1.52	0.81

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	20.44	10.53	0.36	1.37	0.82	0.46	24.78	16.64	0.50	1.64	1.26	0.64
AVG	18.84	14.78	0.66	1.34	1.09	0.70	23.84	19.24	0.68	1.62	1.40	**0.77**
DHondtDO	**25.40**	**20.34**	0.67	**1.64**	**1.34**	0.68	**33.47**	**27.42**	0.69	**2.07**	**1.77**	0.75
EP-FuzzDA	18.56	15.30	**0.73**	1.31	1.09	**0.71**	23.53	19.64	**0.72**	1.60	1.39	0.76

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	18.52	7.01	0.24	1.25	0.54	0.30	23.76	13.63	0.40	1.56	1.00	0.51
AVG	16.76	11.52	0.53	1.19	0.85	0.57	22.68	16.51	**0.58**	1.52	1.18	0.65
DHondtDO	**21.68**	**15.00**	0.53	**1.40**	**0.99**	0.55	**31.22**	**22.72**	0.57	**1.91**	**1.46**	0.62
EP-FuzzDA	16.67	11.28	**0.54**	1.17	0.85	**0.58**	22.57	16.24	0.57	1.51	1.18	**0.65**

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	17.70	5.10	0.17	1.18	0.38	0.22	22.75	11.72	0.34	1.49	0.84	0.43
AVG	15.83	9.56	**0.44**	1.12	0.71	**0.48**	21.67	14.67	**0.51**	1.46	1.04	**0.57**
DHondtDO	**19.82**	**11.75**	0.42	**1.28**	**0.80**	0.46	**29.32**	**19.45**	0.49	**1.80**	**1.24**	0.54
EP-FuzzDA	15.88	8.72	0.40	1.10	0.69	0.47	21.65	14.06	0.48	1.45	1.03	0.56

Table A.9: Results of offline **long-term** evaluation on **KGRec** dataset. The best results are in bold, second-best are underscored.

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	242.2	210.3	0.77	2.26	2.17	**0.93**	242.4	214.6	0.79	2.26	2.19	**0.93**
AVG	239.7	220.6	0.86	2.25	2.13	0.90	240.5	224.0	0.88	2.26	2.15	0.91
DHondtDO	**256.2**	**233.5**	0.84	**2.37**	**2.24**	0.90	**257.3**	**237.8**	0.86	**2.38**	**2.26**	0.91
EP-FuzzDA	235.5	227.2	**0.93**	2.23	1.99	0.82	236.9	230.1	**0.95**	2.24	2.03	0.84

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	237.1	202.2	0.74	2.19	2.07	**0.90**	236.8	206.6	0.77	2.21	2.09	**0.90**
AVG	234.9	212.1	0.83	2.19	2.02	0.86	235.1	215.3	0.85	2.21	2.04	0.87
DHondtDO	**249.3**	**223.2**	0.82	**2.29**	**2.11**	0.86	**249.9**	**227.2**	0.84	**2.31**	**2.14**	0.87
EP-FuzzDA	231.6	217.5	**0.89**	2.17	1.89	0.79	232.0	220.8	**0.91**	2.19	1.93	0.81

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	230.0	194.1	0.73	2.13	1.93	**0.83**	231.5	199.1	0.75	2.13	1.93	**0.83**
AVG	228.4	202.3	0.81	2.13	1.88	0.80	230.3	206.9	0.83	2.13	1.89	0.81
DHondtDO	**240.8**	**211.7**	0.79	**2.22**	**1.96**	0.80	**243.1**	**217.3**	0.81	**2.22**	**1.96**	0.81
EP-FuzzDA	225.9	205.7	**0.85**	2.12	1.79	0.75	227.9	210.6	**0.87**	2.11	1.80	0.76

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	225.9	186.7	0.70	2.09	1.83	**0.78**	227.9	193.1	0.73	2.08	1.83	**0.79**
AVG	224.6	194.5	0.78	2.09	1.79	0.76	226.8	200.3	0.80	2.08	1.80	0.77
DHondtDO	**235.6**	**202.9**	0.77	**2.17**	**1.86**	0.76	**238.5**	**209.8**	0.79	**2.16**	**1.86**	0.77
EP-FuzzDA	222.3	196.7	**0.81**	2.08	1.71	0.72	224.7	202.7	**0.83**	2.07	1.72	0.73

Table A.10: Results of offline **long-term** evaluation on **MovieLens1M** dataset. The best results are in bold, second-best are underscored.

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	**348.9**	224.3	0.49	1.93	1.72	0.80	348.5	230.5	0.51	1.93	1.73	0.81
AVG	317.8	268.8	0.75	1.91	1.70	**0.81**	320.8	274.9	0.76	1.91	1.71	**0.82**
DHondtDO	345.6	**291.2**	0.75	**2.04**	**1.77**	0.78	**349.5**	**298.2**	0.76	**2.03**	**1.78**	0.80
EP-FuzzDA	313.5	278.4	**0.83**	1.89	1.53	0.69	315.2	285.8	**0.85**	1.89	1.54	0.70

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	**323.9**	203.8	0.44	1.82	**1.60**	**0.78**	**330.4**	213.3	0.46	1.80	**1.59**	**0.78**
AVG	296.9	241.4	0.70	1.81	1.54	0.76	305.7	254.3	0.72	1.79	1.54	0.77
DHondtDO	319.3	**258.1**	0.71	**1.91**	1.58	0.73	330.3	**273.5**	0.73	**1.90**	1.58	0.74
EP-FuzzDA	296.4	247.0	**0.74**	1.80	1.42	0.67	303.6	261.2	**0.77**	1.78	1.41	0.67

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	**302.3**	187.9	0.39	1.68	**1.42**	**0.72**	**303.2**	193.5	0.42	1.67	**1.43**	**0.74**
AVG	279.0	218.5	0.66	1.68	1.33	0.66	281.6	226.0	0.68	1.67	1.34	0.68
DHondtDO	296.8	**230.9**	**0.66**	**1.76**	1.33	0.63	300.1	**239.8**	**0.69**	**1.75**	1.35	0.65
EP-FuzzDA	279.8	220.1	0.65	1.68	1.24	0.60	281.4	228.2	0.69	1.67	1.25	0.62

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	**285.0**	175.9	0.37	1.61	**1.31**	**0.68**	**288.3**	182.7	0.40	1.59	**1.31**	**0.69**
AVG	265.1	202.0	0.63	1.60	1.20	0.60	270.5	211.3	0.65	1.59	1.22	0.63
DHondtDO	278.5	**211.5**	**0.64**	**1.67**	1.19	0.57	284.9	**221.9**	**0.66**	**1.65**	1.21	0.59
EP-FuzzDA	265.2	202.4	0.61	1.60	1.12	0.55	270.0	212.0	0.64	1.59	1.14	0.57

Table A.11: Results of offline **long-term** evaluation on **Netflix** dataset. The best results are in bold, second-best are underscored.

	PRS(M=1), group size s=2						PRS(M=4), group size s=2					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	12.50	6.24	0.34	1.21	0.75	0.47	14.56	8.84	0.43	1.43	1.12	0.65
AVG	10.22	8.27	0.63	1.24	1.16	**0.88**	12.87	10.32	0.63	1.44	1.37	**0.90**
DHondtDO	**15.56**	**13.73**	0.72	**1.63**	**1.40**	0.74	**20.71**	**17.73**	0.70	**2.00**	**1.81**	0.81
EP-Fuzz-DA	9.49	8.58	**0.77**	1.21	0.97	0.68	12.23	10.59	**0.73**	1.41	1.21	0.75

	PRS(M=1), group size s=3						PRS(M=4), group size s=3					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	10.96	3.55	0.17	0.97	0.40	0.26	13.69	6.50	0.28	1.22	0.81	0.49
AVG	8.74	5.57	0.42	0.99	0.79	**0.68**	11.95	8.15	0.48	1.23	1.05	**0.75**
DHondtDO	**11.59**	**8.80**	0.52	**1.18**	**0.90**	0.59	**18.40**	**14.17**	0.55	**1.64**	**1.37**	0.68
EP-Fuzz-DA	7.83	5.93	**0.58**	0.96	0.73	0.60	11.31	8.39	**0.56**	1.20	0.98	0.68

	PRS(M=1), group size s=4						PRS(M=4), group size s=4					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	**10.01**	2.08	0.09	0.83	0.23	0.15	12.68	5.07	0.20	1.10	0.62	0.37
AVG	7.85	3.83	0.29	0.84	0.54	0.50	11.13	6.48	0.36	1.10	0.84	**0.61**
DHondtDO	9.29	**5.76**	0.36	**0.92**	**0.61**	0.45	**16.05**	**10.96**	**0.43**	**1.40**	**1.06**	0.55
EP-Fuzz-DA	7.04	4.07	**0.39**	0.81	0.56	**0.52**	10.58	6.64	0.43	1.08	0.82	0.60

	PRS(M=1), group size s=6						PRS(M=4), group size s=6					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	**8.61**	0.77	0.03	0.69	0.08	0.05	11.41	3.30	0.12	0.95	0.39	0.24
AVG	6.67	1.92	0.15	0.68	0.29	0.29	9.99	4.65	0.26	0.95	0.60	**0.46**
DHondtDO	7.00	**2.74**	0.19	**0.69**	0.31	0.26	**13.60**	**7.59**	**0.30**	**1.14**	**0.71**	0.39
EP-Fuzz-DA	6.00	1.96	**0.20**	0.65	**0.31**	**0.33**	9.63	4.60	0.27	0.93	0.60	0.45

	PRS(M=1), group size s=8						PRS(M=4), group size s=8					
	AR			nDCG			AR			nDCG		
	mean	min	M/M	mean	min	M/M	mean	min	M/M	mean	min	M/M
AVG-U	**7.34**	0.23	0.01	**0.57**	0.01	0.00	10.74	2.44	0.09	0.86	0.26	0.15
AVG	5.58	1.03	0.08	0.56	0.16	0.17	9.33	3.69	0.19	0.84	0.44	0.33
DHondtDO	5.43	**1.38**	0.11	0.54	0.14	0.13	**11.94**	**5.53**	**0.21**	**0.97**	**0.50**	0.28
EP-Fuzz-DA	4.95	1.01	**0.11**	0.53	**0.17**	**0.21**	8.97	3.50	0.19	0.82	0.44	**0.34**

Table A.12: Results of offline **long-term** evaluation on **Spotify** dataset. The best results are in bold, second-best are underscored.