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 2022

Dedication.

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: Abstract.

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

Contents

1. Introduction

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 the search results, shopping, listening to music, reading news, to browsing social media, and many more. Recommender systems has become quite literally unavoidable.

We can view recommender systems from a very simple perspective - they are algorithms that recommend items to users. Where items and users can be many different things, items, for example, being movies, news articles, a more complex object, or even entire systems. And users being, real people or other entities that exhibit some sort of preference on which the algorithm can decide.

One of the variants of recommender systems is those where the recommendation result is shared among more users based on their shared (aggregated) preferences. This 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 of the 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. And that's where the group recommenders come in handy.

With groups as the target of a recommendation comes new challenges, among many others, 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 starts to become harder than just simply rating the results based on single feedback, now we have multiple users with possibly very different personal experiences. We want to be fair towards all individuals in the group. But 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 quite recent. With the rise of social dilemmas around recommender systems is the fairness-ensuring topic becoming more important than before. And with that, there is growing popularity towards 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 evaluation of the research without using various methods of data augmentation and artificial data creation. And the definition of fairness is not unified. It can mean many different things and be evaluated with many different methods.

These two aforementioned problems go hand in hand with the very subjective nature of user preference.

Mozna merge obou kapitol dohromady?

1.2 Research objective

We would like to study how fairness can be defined in the context of the recommender systems, how it can be measured and eventually used to improve recommendations in the group setting. And 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. If we could adapt fairness preserving methods such as voting systems from other fields to group recommendation problem. And evaluating 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 and creating a web application in a movie domain that would serve as a platform for online evaluation of group recommender algorithms and provided us with real-user group recommendation data.

1.3 Thesis structure

We start with an introduction to recommender systems and specifically to group recommender systems in the chapter: Recommender systems. Then we will continue with the definitions and evaluation methods for fairness in chapter Fairness. Next, we will introduce few algorithms that are used in the group recommender field in chapter Related work. TODO: check out other works and decide what should be here. This can be nice from the reading perspective, but is it really necessary?

2. Recommender systems

In this chapter, we will briefly introduce in general what recommender systems are (hereinafter referred to as RS) and 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 are trying to suggest items to their users, or from another perspective, they aim to predict how would a user rate (like) an unseen item. They are used in a variety of settings, from 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 systems that aim to deal with selecting a subset from all the items by some filtration criteria, in this case the criteria is the user preference. They become necessary when it comes to suppressing the explosive growth in information on the web and function as a defence system against over loading the user with the vast amount of data that is present in almost every system today.

They can be views 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 of the user and the recommender system, in varying degrees of freedom. From passively accepting the RS recommendations to actively interacting by giving feedback and stating the preferences.

2.1.1 High-level examples

Recommender systems are used in multiple ways, we now present a few 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 user could like based on previous orders or viewed items, either related or alternative choices.
- **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 others they viewed, globally popular for a regional subset of the user-base and so on.
- **Personalized news feed and social media feed.** We want to offer user interesting content that would keep them engaged with the service. In the recent years there is push towards more social responsible RS design in this context due to the overwhelming power the social media have. It is important to deal with problems such as polarization [3], fairness and disagreement.

- **Expert systems** that help doctors, operators and other people to make an 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 solution with only weakly defined requirements.
- **Search experience**, that takes into consideration 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 that were liked by users who have a similar taste for other items that they both rated. And thus exploiting data of users with similar preferences.
- **Content-based filtering** (CB)
  Uses item features or item descriptions to recommend items similar to those that 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**
  Depends on hand-crafted deep knowledge about items. 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**
  Combines multiple RS, either the same type with different parameters and/or different types together in order 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 is feed as an input to another one.

The popularity of the first two approaches varies from domain to domain. Some domains naturally contain item-specific data which allows using the *content-based filtering* for example product parameters in e-shops, but 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 for both, CF is able to extract latent meaning from the data that would remain inaccessible to CB that relies on items' features. But at the same time, it can cause problems to rely only on user-item interactions because we need a lot of data in order 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*.

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

One additional approach we did not include in the list is *Critique-based recommendation*. It's popularity is quite low, but still worth mentioning. It acts as sort of 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 user either accepts or rejects them. Based on this critique from user we narrow down the user's preferences and offer different (filtered/extended) set of items and try to further guide the user to a satisfactory result. The critique can some cases be provided for not just the whole item but even just the 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 (hereinafter referred to as single user RS). But what do we do, when we have a group of users that we want to recommend to? For example, a group of friends selecting a movie that they want to watch together or a group listening to music?

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. Results of a recommendation for the group do need to reflect and balance individuals' preferences among all members.

2.2.1 Characteristics of group RS

There are many specifics that contrast GRS with single RS. Usually some form of aggregation needs to be performed in order to make the transition from a single user preferences that we gather to the recommendation for an entire group.

Situations differ for small and big groups, where with the increasing amount of users the complexity of the users' preference increases. At the same time, every person is different, someone is more forgiving and someone is less conforming with their taste.

Although, there in not many reported deployments of such systems yet, we see domains which would greatly benefit from the use of GRS.

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

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

2.2.2 Challenges

We now mention some of the most important challenges that are present 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 preferences of individual users together. It is possible to not support groups at all and let users deal with the act of combining them via discussion. But then the problem collapses back to single user setting, where the user is the whole group. Therefore we need to decide how and when to merge them. Main two approaches are mentioned in 2.2.4.

- **Divergent group preferences**
  There exist users that 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 similarity between users. And 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 groups 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 harder due to the possibility that not all members leave a rating. In some cases gathering the 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 account of one person, therefore it is hard to distinguish what are preferences of that one individual and what are 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 when it comes to 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 a priority over the selection. Second example could be that we would possibly want to prioritize satisfaction of individual in the group that were less satisfied the last time an item was consumed.

- **How to explain provided recommendations**
  Explaining single RS is already pretty hard, with algorithms such as collaborative filtering it is hard to explain why we are recommending an item apart from the obvious explanation of '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 harder and harder.

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

- **Not all members present**
  What can be done if we design a GRS for a specific group and some of the members of that groups 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.

  And as a different problem not entirely relevant to our work but still important is that of how do 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 only a single user is selecting an item, lets say a movie to watch, they 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 the implicit feedback of the group.

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

2.2.3 Classification

We now mention some basic classification s found in [14].

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

  Some GRS start with a good knowledge of preference of each member of the group, such as if we have a system on a popular music streaming service, but 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**

  We need to take into a consideration if the GRS provides only a set of items that are further filtered by the group members before they select an item that they like or if we then already present the users with the recommended items, such as a recommendation of a shared group playlist at a social event. We, for example can recommend more controversial item options when the list will be further narrowed down by the group members.

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

  There are situations in which the priority of group members when it comes to satisfaction differs. We can have systems that allow to set different weights to 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 predefined length, a set that does not have an explicit order and other. The output in the case of expert systems can be a graph, set of rules, or a feedback gathering question.

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

  Another difference is with how the resulting recommended items are 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 is the case of playlist generation in the music domain? We need to gather feedback and calculate the possible utility separately in these different 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 the preferences of all of its members. 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 of 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 4.2.1.

In the RS aggregation approach we further distinguish between situations where we have predictions for all possible items and therefore can do aggregation directly on the ratings of all items or if only have a list of recommendation for each user - subset of all the items. 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], but the approaches are very similar, they only differ in the availability of provided results from the underlying RS, so we group them together under one main direction.

Further, both group aware RS and aggregation approaches do both have some advantages and disadvantages. One of 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 aggregating directly the top items provided, or in more involved way by utilizing the predicted ratings. On the other hand, 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 case of the first method, the group aware approach, can potentially be extracted.

We will go in-depth to discuss techniques used in the latest literature in 4.

At the same time, we need to define what does it even mean 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 go in depth to describe common approaches to these problems in 3.

3. Fairness

So far, we have discussed recommender systems and the methods that are used in the field. But 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 it plays when using machine learning models with potentially sensitive data and see how and if we can make group recommenders better when we understand and define the underlying properties.

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.

3.1 General

The word fairness itself is very 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 from the 1960s as we can see on Figure 3.1.

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 a 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].

And 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 you care about someone having more if you 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. Where humans are willing to seek equalization of outcomes even if it means that they will loose 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 the fairness itself - "It is not fair if someone else gets something we don't.". 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 in some cases can lead even to larger envy involved.

In some cases we observe a pattern of generosity, where children are willing

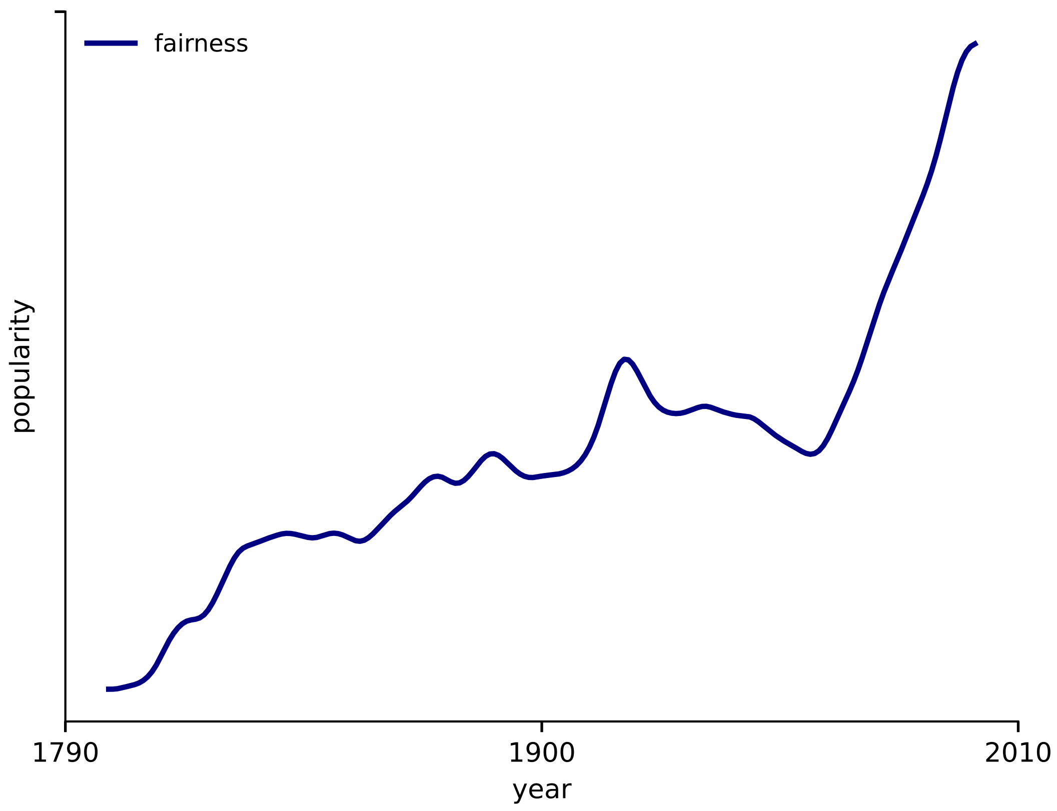


Figure 3.1: The graph shows the phrase ”fairness” occurring in the corpus of English books from 1800 to 2010. Source: Google Ngram Viewer, corpora 2012. [19]

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 out 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 as one of the main problems of today’s world. But discussing the social aspect of fairness is beyond the scope of this thesis.

When it comes to the true nature of fairness on the deepest level, it could even be possible that the notion of fairness emerges together with cooperation and therefore language and communication. It would mean that it is an inherent property of any intelligent agent that is 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 and that leads to a higher stability of the group which would give an evolutionary advantage. But more research has to be done, as our understanding of intelligence, consciousness, and related hard to quantify phenomena is lacking.

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

3.2 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 main goal of defining fairness for the group RS sub-domain, but we think that it is very important topic in general, deserves more spotlight and can be a good middle step to understand specifics of the group RS setting.

3.2.1 Sources of algorithmic unfairness

How can a computer that has 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 that is used to train those algorithms come from the real world, where bias and unfairness is unfortunately still present. And so, 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 their class, race, political views and other factors which we, in the modern democratic world, consider as protected(sensitive) characteristics, which are not to be used as discrimination against an individual. But this type of bias in those circumstances is a knowingly designed and required feature of the system and therefore we will not discuss these systems any further.

In a more rigorous way, we can say that output of any machine learning (ML) algorithm is usually just a product of the underlying data. By design, accurate classifier will reproduce patterns found in the training data. And 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 3.2.2:

- **Biases already included in the data set**
  Such as dependence/correlation of data based on sensitive characteristics. Bias found in the data is by design reproduced by an accurate classifier.
- **Biases caused by missing data**
  Missing or filtering out some of the training data can results in a data set that does not represent the target population.
- **Biases stemming from algorithmic objectives**
  While training, we usually minimize some error, but that can lead to prioritization of interests of majority 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 having a larger entropy.
- **Biases caused by ”proxy” attributes**
  Some attributes that are not directly considered sensitive still can contain information from sensitive attributes. In other words, they are not independent. The algorithm can therefore use the ”proxy” attribute and exploit the sensitive attribute indirectly.

It is important to define which sensitive characteristics need to be taken into account. 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 impacts of algorithms on society. We observe bias to specific groups of 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. We are, after all, are more than ever involved and surrounded by technology. It is therefore essential to understand the effects which 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 one.

3.2.2 Examples of algorithmic bias

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

- **Amazon’s automatic recruiting system**
  As reported by [27], in 2018 it was found that new system for hiring people

for Amazon was biased towards women. IT field being mostly male dominated - women represent only around 23% as stated in [28]. Due to this disproportion the algorithm discovered in training data pattern between gender of the candidate and hiring results which lead 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 other. This bias is a combination of "bias already included in the data set" 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 own credit card in 2019. It works as follows: after the sign up, the user receives certain credit limit by the service provider (Goldman Sachs). Some people, as reported in [29], noticed that their wives were assigned smaller credit limit even though their credit score was higher and they only had one shared bank account. In this case, investigation by New York State Department of Financial Services came to a conclusion based on an extensive analysis that no unlawful discrimination against applicants have taken place.

- **COMPAS** - Correctional Offender Management Profiling for Alternative Sanctions
  COMPAS is an algorithm that was used in US justice system to predict the likelihood that a defendant would become a recidivist. Analysis [30] found that black defendants were often predicted as being higher risk than they actually were and on the contrary white defendants were predicted to be less risky than in reality. In case of re-offended, this predicted risk was almost twice as high for blacks compared to whites. They conclude that the tool is very imprecise and does not reflect the true likelihood that it was designed to predict.

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

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

3.2.3 Measures of algorithmic fairness

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

At first sight our idea could be to just remove features that we consider sensitive entirely from the data set, 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 to be predicted with measurable accuracy, but this balance can tip over if we put many of these slightly correlated features together. 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:

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

  Meaning that the probability of the given prediction is 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 to 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 dependence of 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 to 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.

3.2.4 Outcome vs. opportunity

From separation and sufficiency, we see that they 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 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 who to hire towards any of the gender. Preferably, we don't 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 great deal of harm, while implemented incorrectly.

With the already discussed topics in mind, we can get back and 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 balance of preference between members of a group. 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.

3.2.5 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 on the data path is the change to the machine learning process made. 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 independence of the sensitive attributes. But as with any data transformation, that change data properties and distributions, we need to be careful as not to hinder the efficacy of the final model.

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

  This type of change seems to be the hardest to technically and algorithmically implement. We need to have an access to the whole collection of raw data in order to ensure that we are not biased. Further more, we limit ourselves to ML methods that allow this type of constrain modulation. On the other hand 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 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 evaluation of the model performance harder.

3.2.6 Other adverse effects

It takes a lot of time to make data sets and models unbiased, especially so, because bias is included in most of the data sets 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 in 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 it's decisions (after being utilized in the environment from which we gather the data set) will consume its own (previous versions') decisions as training data. This can lead to amplification of biases that were already present and to further skew the distribution of the underlying data. This effect is called "negative feedback loop".

In order to build robust and fair algorithmic systems, we need to 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 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]. Lets 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 lets again assume that in the prior data the conversion rate of 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 the page X, and not calling to 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. Which directly leads us to a second adverse effect of echo chambers.

Echo chambers

The feedback loop described in previous section can be observed in effect not only when retraining algorithms, but in social groups as well. People naturally seek out information that reinforce and support their existing views as stated in [33]. Serving content to user that supports 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 user spent interacting with the service and return rate of the visitors. We use them as a target for the optimization and that leads to a creation systems that naturally locks 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].

3.3 Fairness in Group recommender systems

So far, we have discussed algorithmic fairness in the context of equality of opportunity, where the main goal is to not discriminate against an individual. The same issues are present in the Group RS domain, but we will focus on equality of outcome more. 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-likeable, 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-likeable, if it is was 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 settings, as we will discuss later, will member-likeability a better choice, in others it will be the group-likeability. We can view it as an opposing force. We can either push towards more or less uniformity.

3.3.1 Specific cases of fairness

Let us now mention some of the main ways and their differences, as 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 to meet due to the differences in members' taste. We can view preference of a group more like a set intersection than set union.

- **Fairness in a list of items that are consumed sequentially** This setting differs from the previous, because we can be recommending 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 by 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 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.

3.3.2 Evaluation

In order to asses performance of group recommender systems, we need to define some metrics as 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 an information about the effect of that item to the users utility function.

The utility function we want to calculate is the total relevance of the recommended list to each user and we consider fair if it is balanced among users, where no one is systematically biased against. We therefore need 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 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 as how to aggregate relevance of items in the recommended list. We will mention further specifics about evaluation in 7.

3.4 Other criteria

We have discussed fairness as one of the criterion for 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 will be 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 optimise 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.

3.4.1 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 that they have no expertise in. 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.

3.4.2 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 data-set too well. Some models are explicitly based on similarity between users. A great example can be user-based collaborative methods from recommenders systems that were described in 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 the 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 data-sets to predictions of the models can be the gathering and usage of massive training data-sets it self. Fair use which is vaguely mentioned by almost all service providers that gather users data should be revised and brought up to standards of the 21st century.

Another dangerous privacy breach can be an identification of users from the published data-sets. 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 person's movie watch history, for example extracted from a public source such as IMDb.com, we can match the data from Netflix challenge to this public source. The Netflix challenge was later canceled.

3.4.3 Trust

Different machine learning applications usually require 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 should a person 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 effectivity of the system can diminish even if the training and testing data tells otherwise. Trust can be greatly increased when providing explanations accompanying the output of the system.

3.4.4 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 a 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 Amazon book store on Figure 3.2 and second from Facebook's explanation of why was a particular advertisement served on Figure 3.3.

Explanations can achieve not only increased trust in the system as mentioned before, but 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 example are neural networks. Neural nets keep context in neural connections which are very hard to provide explanation for. We sometimes refer to this property of a system that we do not fully understand as "black-box" systems. We know how they work, how to train them, but due to the inner complexity we fail to explain the exact way of 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].

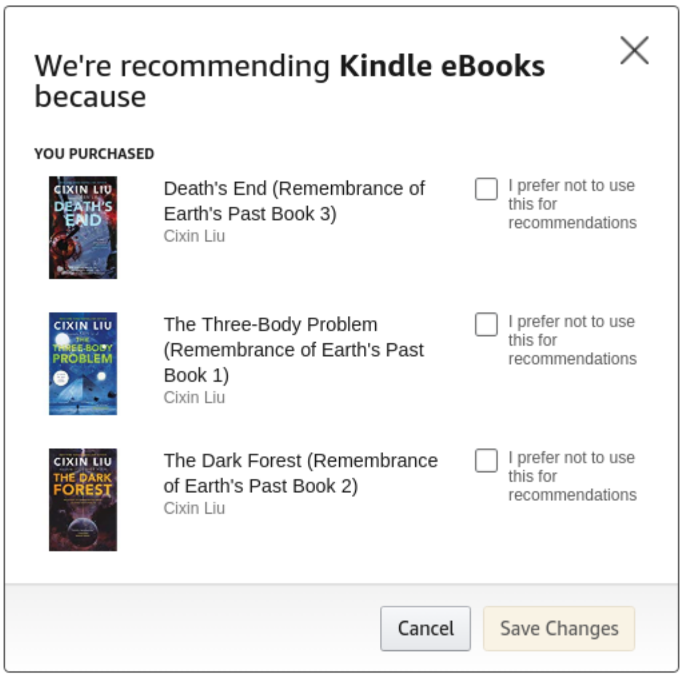
Explainability can be even indispensable. Lets for assume that a ML model is used to recommend a legal action against an individual in a judiciary setting. We cannot soundly present it as an evidence without being able to justify its actions and warrant its correctness.

3.4.5 Legitimacy

Legitimacy can be viewed as an another step after explainability. When we provide a prediction together with a sound reasoning that can be verified either by a human analyst or by an other 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 analysis, if set up correctly. But as mentioned many times before, we need to be very careful with the data that we use for training. Lot of care needs to be taken even if we do not use an automated training solutions. Most of the current systems used in the mentioned legal domain and other systems where legitimacy is required are most probably base on hand crafted rules that can very well be subject to bias as well. One of the examples being the system COMPAS presented in 3.2.2.

3.4.6 Consistency

For some systems it is very important to stay consistent in outputs even with a slightly different inputs. Lets assume, that we are developing an illness detection


We're recommending **Kindle eBooks** because

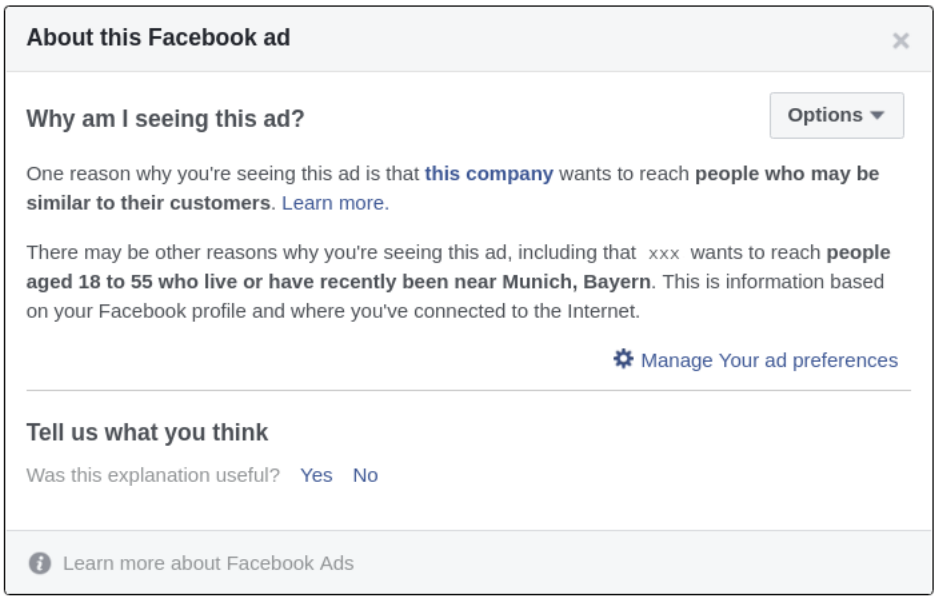
YOU PURCHASED

Death's End (Remembrance of Earth's Past Book 3)
Cixin Liu
I prefer not to use this for recommendations

The Three-Body Problem (Remembrance of Earth's Past Book 1)
Cixin Liu
I prefer not to use this for recommendations

The Dark Forest (Remembrance of Earth's Past Book 2)
Cixin Liu
I prefer not to use this for recommendations

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


About this Facebook ad

Why am I seeing this ad?

One reason why you're seeing this ad is that **this company** wants to reach **people who may be similar to their customers**. Learn more.

There may be other reasons why you're seeing this ad, including that xxx wants to reach **people aged 18 to 55 who live or have recently been near Munich, Bayern**. This is information based on your Facebook profile and where you've connected to the Internet.

Tell us what you think

Was this explanation useful? Yes No

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

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. Driver-less cars that abruptly change direction with only one pixel change in the input would not induce trust of its users.

From a different perspective we can view fairness as a sort consistency in the way that protected attributes of people do not have an effect on 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 it self. If a user is browsing a web page, lets 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 it will affect the user experience while listing and using the provided information.

3.4.7 Novelty

Novelty can have multiple meanings when we use it to describe RS. Firstly it can be that the item it self is a new addition to the data-set 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 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 items 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 that an unusual item is presented to the user. Such situation can occur either due to the RS incorrectly assuming that the user will like that item, or by, 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' preference, which needs to be done by gathering feedback on items outside of the users current preference model. Balance between exploration and exploitation is difficult, but 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 for in domains where exploration is important.

3.4.8 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 comparison for the recommendation. Then this item will be left out and never recommended.

Other coverage problem is with items that are popular. 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 a same type of phenomenon as Negative feedback loop mentioned in 3.2.6. It then happens that item exposure distribution will be unnaturally skewed even more towards the popular items. This can lead to decrease in system's performance due to some items that are less popular not being recommended even if being possibly a better choice for 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.

3.4.9 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 as trueness) is a measure of how far away outputs of a 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 type of the systems outputs. We can see an illustration for continuous value prediction on 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 act as a FP label.

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

3.4.10 Performance and scalability

We can design any algorithm we want, but what for will it be if 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 performance and with it related property of scalability.

Performance of as system can be described either in a form of total computational resources needed for processing, either the whole deployment or normalized to a single user. Other possible performance property can be the length of time needed for 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 the 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 is the case in some RS algorithm, we will be growing the system requirements most probably linearly with the amount of users. This can be fine in for example research applications, but becomes a real concern when applying to a world wide applications used by millions of users. At these scales we will be required to scale using not only vertical scaling (that is using a 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. First, specifying the main approaches, dividing them into groups by how they approach the transition from single-user preferences to group results. Then mentioning the main simple techniques and describing how they interact with the common recommendation approaches. And at last diving into some of the more advanced methods that optimize the aggregation in a more elaborate way.

4.1 Categories

We assume that we have individual user preferences (if group preferences are available, then the task actually 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 individual 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 member of the group into a single set of preferences that can be then directly consumed by a recommender system, therefore creating a group preference model. Aggregating the single-user preferences either directly by aggregating ratings of seen, or and 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 on Figure 4.1.

- **Aggregate predictions**
  Aggregates predictions outputted from RS. Recommender recommends separately for each member of the group based solely on their single-user preferences. Then the resulting recommended items are aggregated together to 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 common utility function from all recommended items and select those that are the most fitting to the group based on this utility function. We will discuss both in more detail in 4.2.

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

- **Aggregation is a uniform part of the recommender**
  In this case, the algorithm directly works with users of the group 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. Some could say that for example aggregating the user profiles in matrix factorization makes the aggregation inseparable, because there is a specific preprocessing done before the aggregation step. But 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 for themselves 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 main literature [17] and [2], where they also make a distinction into three groups, but instead based on the data that the aggregation processes and the position. Instead we have chosen a little different distinction based more on interaction of the aggregation with the recommender system, essentially putting two groups from aforementioned literature under single group (*aggregate models*), but with the mentioned possible subdivision.

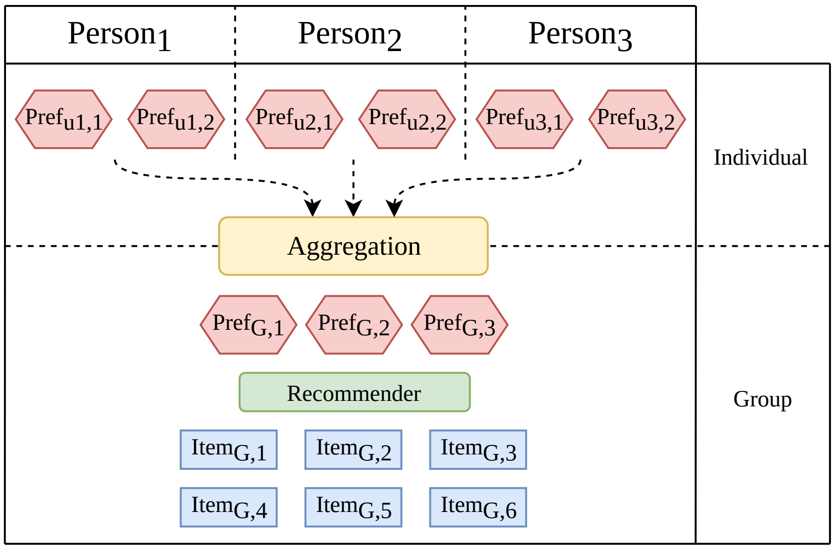


Figure 4.1: High level overview of group recommendation with aggregation of individuals' preferences, before recommendation.

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 2.1.2

4.2.1 Methods

The main aggregation methods can be divided into three groups, based on their high-level approach into - majority, consensus, and borderline based strategies.

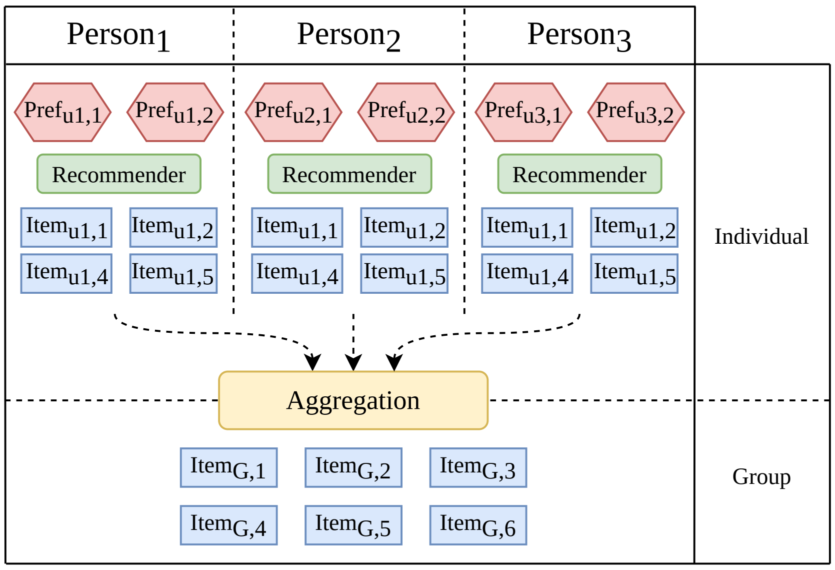


Figure 4.2: High level overview of group recommendation with aggregation on top of recommendation results for individual users.

The majority-based generally use the most popular item among the group members, the consensus-based consider preferences of all the group members, and the borderline-based consider 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

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

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

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

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

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

- **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

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

- **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 location of the item in this ordered list. Worst item receiving 1 point and 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:

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

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

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

- **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 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.

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

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

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

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

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

- **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, which is that 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. Where after the top item is selected we reset the calculation, therefore selecting top items in an iterative manner. 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 a recommendation in order to provide a full group recommender system, which will serve as an introduction of the inner workings of such system in order for us to then 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 using them as *prediction aggregation* after the recommendation is quite 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. But in the following section we will assume that we have a full access to the rating data of the outputted items. We further assume that if necessary we can acquire ratings and ordering for all possible items.

As shown on 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 4.2.1.

Model aggregation

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

First, the aggregation of ratings, in other words creating a set of ratings that represent the whole group. This approach is preferred due to its simplicity. We can use many of the before mentioned simple methods, mainly the *consensus-based* which aggregate ratings directly (positive as well as negative ratings). Other mentioned methods can be used as well, but 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. Where we aggregate not the rating themselves 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 little 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 quite straightforward, they have mostly very simple and clear objectives, and try to optimize in different ways some quite vaguely specified goal. But as we have discussed in 3, the objective is quite hard to define and measure. We will now introduce some 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 a 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.

Introduction

As mentioned they focus on the *fairness* of top-N_G (*fairness* of the top N recommended items for the 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 2.1 the difference between optimization independently item after item versus in some way for the whole list can yield significantly better results when it comes to sequential consumption of items or and balancing users with different preferences. For example, being one member of the group being treated unfairly due to different preferences from the rest of the group, then fairness defined as a property of the whole list can in some way compensate and give more priority for balancing fairness forwards this one user. This kind of unfairness is the main problem that they try to solve.

We have already mentioned one algorithm that tries to balance fairness of the whole list in 4.2.1, FAI, which generates the aggregated results in iterations, each iteration optimizing for a different user in turns. This approach solves the aforementioned problem in a very simple (although somewhat dull) way by not considering the group preferences at all and just trying to please everyone in turns not considering if the item will be liked by the rest of the group.

Approach

What if we have instead taken into account the previously recommended items when we are selecting the next one in the list? The authors propose defining *fairness* together with ordering within the group by saying that "a top-N is fair to a group if the relevance of the items is balanced across the group members for *each prefix* of the top-N_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:

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

Where 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). 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 (#items $-$ 1) points.

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

$$p(\text{rel}|u,i) = \frac{\text{Borda-rel}(u,i)}{\sum_{j\in\text{top}-N_u} \text{Borda-rel}(u,j)} \tag{4.15}$$

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 atleast one item from set S is relevant to the user u as:

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

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 the 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 item. 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. And 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 Package-to-Group Recommendations

Paper [46]?

4.3.3 Top-N group RS with fairness

Paper [47]?

5. Datasets

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

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

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 will describe in the single user datasets subchapter.

And finally, we describe a data transformation library that we create for the purpose of simplifying the research efforts in the RS domain and for evaluation of our proposed algorithm.

5.1 Single user datasets

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

If we talk about 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 about the data format transformations follow in 5.1.6.

5.1.1 Movie Lens

One of the most well known dataset 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 recommendation by researches at University of Minnesota.

Dataset is in a good format that is quite easy to parse and use. Further description follows in 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 datatables: 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 GFAR method introduced in [45] and described in 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 description using text and tags that were created by the person that 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 at all. We have made this decision to simplify comparisons and 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 'Mechanical clock movement' sound, 'Industrial elevator' sound and so on. The need for these sounds are most probably driven from people that are 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 [48], where they were introduced, they are altered for the needs of research in Recommendation Knowledge Graphs.

Further, the original data that were used for creation of this datasets are described in [49].

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 datatables: 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*, open competition with main prize of 1 million dollars. It contains data of 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 since.

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 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 datatables: Ratings (detail in Table 5.4), Movies (detail in Table 5.5)

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. Main purpose of this dataset was to study and develop better algorithms for automatic playlist continuation where the system would be

[1]Number of items, users and user-item interactions are in order - music dataset; sound dataset

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.

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.

able to recommend songs that are similar to those that are 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 are playlists and not users, we utilize a different view on the dataset, where each playlist will represent a single user. This way, we have an another big and organic dataset at our disposal. It can therefore be used not only for playlist continuation tasks, but for the classical RS domain tasks as well. In a sense, a single playlist is a specific subset of preference of the user that have 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. But 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.

`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.)

Number of items: 2,262,292
Number of users: 1,000,000
Number of user-item interactions: 66,346,428
User-item interactions format: one-valued implicit feedback - user added a song to a playlist
List of datatables: Tracks (detail in Table 5.7), Ratings (detail in Table 5.6)

5.1.5 Comparison of datasets

We have described each dataset separately, let us now compare them together to see how they differ. As we can see on Figure 5.1 the Spotify and Netflix datasets are the biggest. We see that KGRec dataset is almost two orders of magnitude smaller than the rest of the datasets. As already mentioned, we have selected it due to it being utilized in the related literature. Spotify dataset is different by having two orders of magnitude more items, which can present a challenge on its own. If we for example 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.

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.

On Figure 5.2 we present the distribution of ratings among users. The left y-axis, blue, present count of users with particular number of ratings. We have clipped the users with high number of ratings so that we can better see the interesting part of the data, which would otherwise be squashed by the long tails. 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 visible on the figure as well and it is most probably the reason why we do not see the same initial rise in ratings as for Netflix and Spotify datasets'.

KGRec dataset is different, it is more jagged due to the smaller effect of being smoothed out by the the amount of data as with the other datasets. Interestingly enough, by contrast to the other datasets, the number of users does not follow an exponential distribution. At first, we thought the reason is, that the dataset was

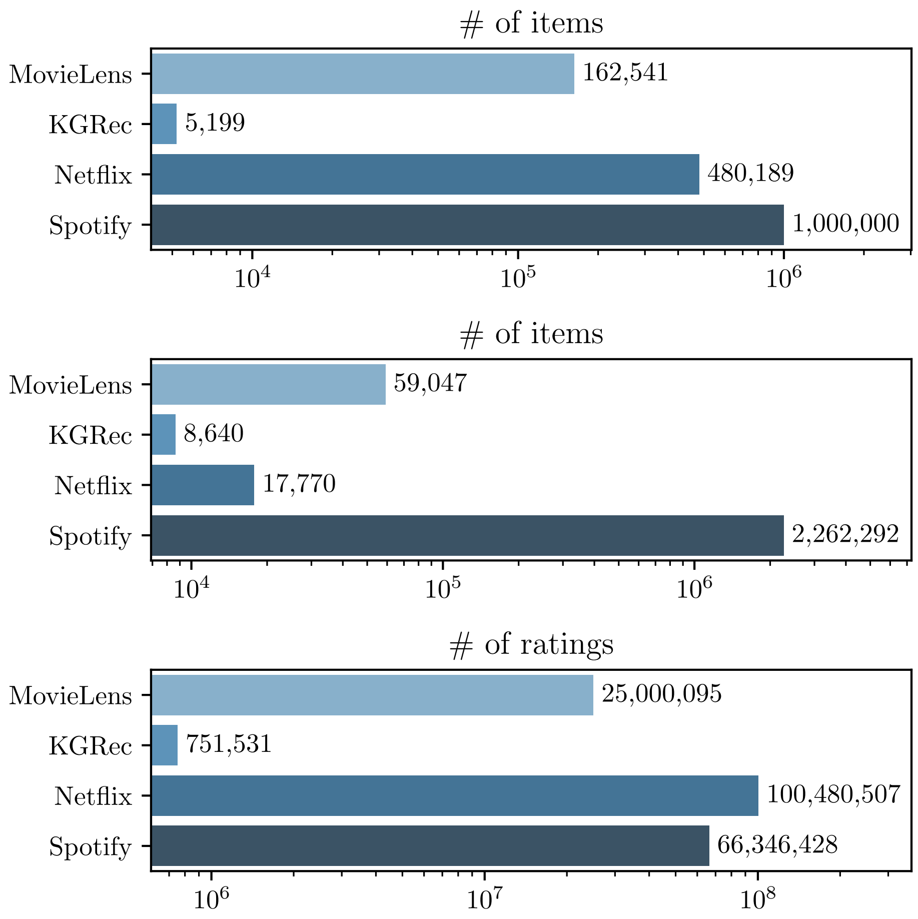


Figure 5.1: Size comparison of the selected datasets. All x-axes are log scale due to the big differences between the dataset.

gathered at a website where users listen and download songs. These songs are always a part of an album and a user is downloading one song from an album, then they will most likely download the rest of the album as well. But, that would not explain why this number of ratings per user starts at around 75.

The true reason for this can be partially found in the original paper - [48]. As already described in 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 10 interactions were removed, users with less than 50 item interactions were removed and only songs with over average plays were counted as a user-item interaction. But this would not explain the non exponential nature of the distribution. We have downloaded and explored the original dataset from [49], the original dataset is not only a one-valued implicit feedback but it is the number of times a user has played the song. When we visualise the original dataset using 'sum of plays per user' instead of 'count of interactions per user', then 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 amount of songs multiple times which then creates more of a normal looking distribution.

Further, Netflix dataset looks similar to Movie Lens with two exceptions. First being that Movie Lens includes only users with at least 20 ratings, therefore the

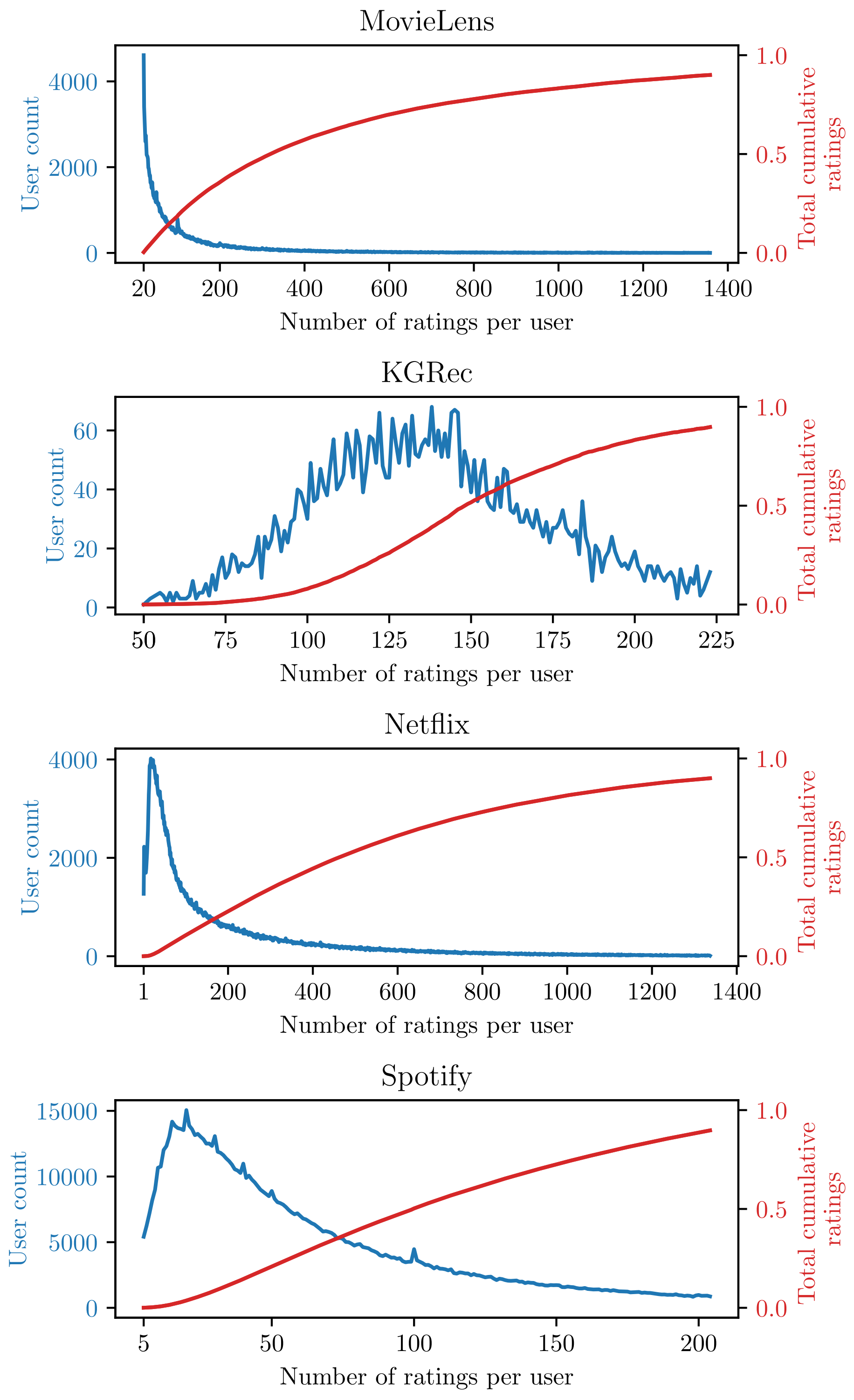


Figure 5.2: Distribution of ratings for groups of users by how many ratings they have made. Blue line shows how many users have made that particular amount of ratings, while red line shows the total cumulative mass distribution of ratings. We have clipped the number of ratings by total cumulative ratings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

initial increase in 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 is more ratings among users that were more active on the service and interacted with many items.

5.1.6 Datasets gathering and processing

eProcessing the above mentioned datasets was more difficult than it should have been. They are not easily accessible, some of them 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 have a shared storage where they would be available in already transformed form, but that is not feasible due to datasets' licences. Let us now describe data transformations that we have done for each of the datasets. We aim to have the datasets in standard zipped CSV format that can be easily loaded by most of the popular data manipulation tools such as Pandas. Additionally, if not misleading, we prefer the columns of the datasets to be named in unified fashion.

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

- **KGRec** dataset is available for download at the authors web page upf.edu/web/mtg/kgrec. Unfortunately, downloading the dataset is not straightforward due to the download link being 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 items ids to sequential values was performed.

- **Netflix** dataset is available at an independent web page kaggle.com/datasets/netflix-inc/netflix-prize-data. The original web page of the challenge is no longer available. The dataset was additionally processed by the uploader by aggregating the small per movie rating files into four bigger files. This dataset was in 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 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 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 easily download the available datasets. Further description of this tool and where it can be downloaded follows at the end of this chapter in 5.4.

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

First, we would like to point out a critical flaw that most literature in our domain has. If 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. Same can sometimes be said about papers that use single user datasets with synthetically generated group information. Some of the time, it is not entirely clear as how have the synthetic groups been generated and what values have been the parameters set to.

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

	#users	#items	#groups	avg. g. size
CAMRa2011[50]	602	7,710	290	?
Douban[51]	23,032	10,039	58,983	4.2
Gowalla[51]	25,405	38,306	44,565	2.8
Mafengwo[50]	5,275	1,513	995	?
Meetup[52]	42,747	2,705	13,390	16.66
Plancast[53]	41,065	13,514	25,447	12.01
Yelp[51]	7,037	7,671	10,214	6.8
Weeplaces[51]	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

CAMRa2011 dataset was released in 2011 for 2011 ACM Recommender Systems Conference. The dataset is unavailable at the original location and we were unable to retrieve it from the web. Numbers in Table 5.8 are taken from [50]. We have found altered version of the dataset at github.com/LianHaiMiao/Attentive-Group-Recommendation, which is a GitHub repository for the code and experiments from [50]. The dataset is quite small and we are doubtful about its quality and source.

Douban

This dataset has similar problems as the CAMRa2011 dataset, we are unable to retrieve it from the web. At the same time, in the already mentioned repository for [50], 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 has been dissolved in 2012 when it was acquired by Facebook. The dataset contains people logging in locations that they have visited. The dataset can be easily downloaded at yongliu.org/datasets/, we have discovered this page and dataset in [51].

This dataset does not directly contain any group information but it could be inferred by combining the check-in data in of format '<userid, placeid, datetime>' and friendship data that link pairs of users '<userid1, userid2>'. If you and some of your friends visit the same place at around the same time, we can state that probably you were part of the same group. And if you visit more places together then the chance of a random occurrence drops significantly.

We suspect that due to the data being location based there will be a substantial location similarity bias where if you visit a certain place then you 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

Very small, proprietary dataset mentioned in [50]. Dataset is unavailable for download. We were unable to find other source from where we would be able to download this dataset.

Meetup

This dataset is crawled data from website meetup.com from [52] used in [53]. Meetup is a popular web page for meeting other people with same hobbies and interests. This dataset contains only data from two regions, New York City and 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, you 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 [53] where this dataset is mentioned, there is no download link, only a reference to [54] where no additional information about the source is provided.

Yelp

This dataset contains reviews for businesses and places. In [51] 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 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 or Facebook. It contains information about check-ins and friendships, same as the Gowalla dataset. It can be downloaded from yongliu.org/datasets/. We have discovered this page and dataset in [51]. Same arguments for the group information and location based bias hold for this dataset, again, same as for Gowalla dataset.

How are groups created is not described in the original paper. Other information which is missing is a full description of how was the raw dataset transformed and filtered. 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 subset of the mentioned datasets for further analysis and for inclusion to 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 'poor' rating if the dataset cannot be downloaded at all. 'ok' rating if we can download the dataset with some difficulties from either the source in the mentioned paper or any of the related linked papers, we well as if we can download the dataset from an unrelated source. We award '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 'poor' rating if the group information is either not very fitting to our use case, if 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 the quality is good or great respectively. Ideal situation is if the dataset contains a 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 isn't 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 not well known service of 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 around the world.

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

With all relevant information about each dataset that can be 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 with 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, 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.

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

We are unable to enrich the datasets with group-item interactions, because that is the task we are aiming to solve. We would in a way just cross evaluated a group recommender system with another one which 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 to create 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 in the public transportation or dining in a restaurant, we would observe a very wide spectrum of preference 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 similarity, 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 creation of the 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.

Other argument against the use of 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 already discussed in 3.2.

Using user-item interactions for measuring similarity is very convenient due to this information being the most accessible. It is very similar to a user based collaborative filtering discussed in 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 amount 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 [55], 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 vectors 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 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. Originally, we wanted to select Pearson Correlation Coefficient based on the before mentioned stability, but interestingly, PCC and the Cosine similarity are overlapping substantially. This is due to the ratings being very sparse which causes the means in PCC calculations to be close to 0 which then corresponds to the Cosine similarity. Further, PCC and Cosine similarity on average have more samples with higher similarity and less 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 how to determine a similarity of two users. Further, we need to have a way of how to modulate the amount of similarity in the group. There is many interesting ways as how to create synthetic groups. Let us now present some of them.

Some of possible types of group creation:

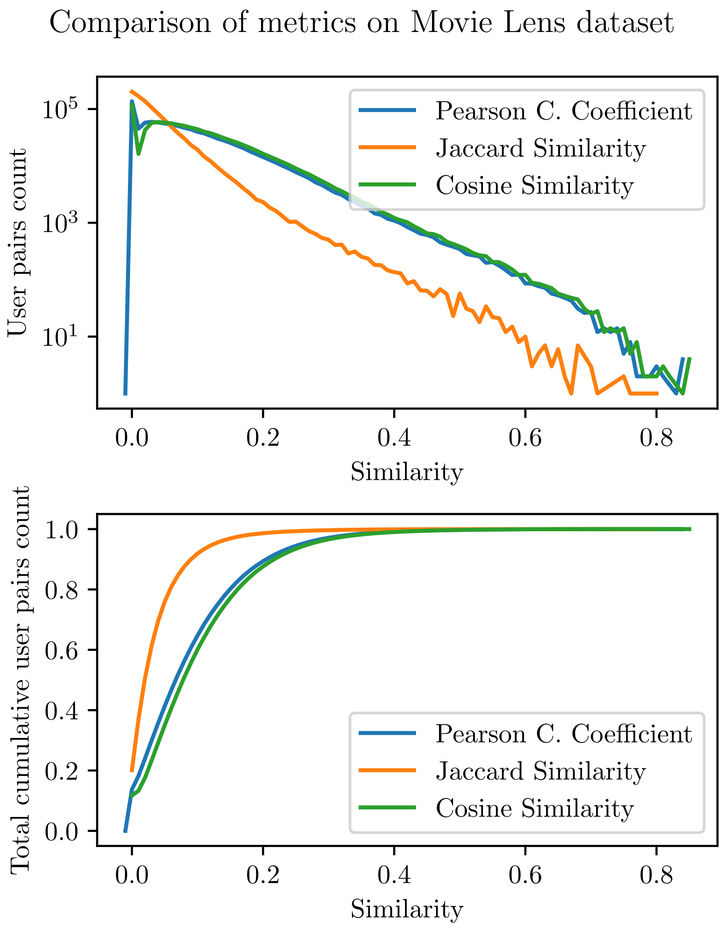


Figure 5.3: Comparison of similarity metrics on Movie Lens dataset.

- **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 neighbours algorithm. Problem with this approach is that very rarely 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 similarity between the pivot and every other user in the dataset. Which is not ideal for the sizes 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 neighbours 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.

  Other idea would be to select the next candidate based on the similarity to the last selected user instead of the pivot. In a way creating a chain of users based on the similarity of the individual links(users). This way, we could create more variable preference 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 either from the similarity or from the ordering of all the candidate members.

- **Similar to the whole group:** Instead of selecting candidates to be similar to a single main user, we can select next 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 the bigger possible preference overlap. But it has a disadvantage of needing to perform more computation of similarity after each individual member selection.
- **Dissimilar:** Apart from finding group members based 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 the next user which is as least similar as possible. Next, instead of selecting another user in the same way, we can calculate dissimilarity of all other uses with the joined preference of the users that are already selected to 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.

Easiest and most convenient option would be to calculate the similarity matrix for all users and then select groups based on this single calculation. But, this matrix can become pretty huge with the growing amount of users. Lets assume our worst case of 1,000,000 users of the Spotify dataset. Then we will have $10^{12}/2$ of similarity calculations that need to be processed and stored. This becomes difficult to manage as the amount of memory needed would be around on the order 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 that situation is somewhat unlikely and in total we will save orders of magnitude worth of similarity calculation (potentially depends on the number of groups we want to generate).

At random

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 amount of users at random 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 a pressure for 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 would like to select a candidate with a probability that corresponds to their similarity with the pivot user and decreases quite heavily to still ensure there is enough of a push towards selecting similar users.

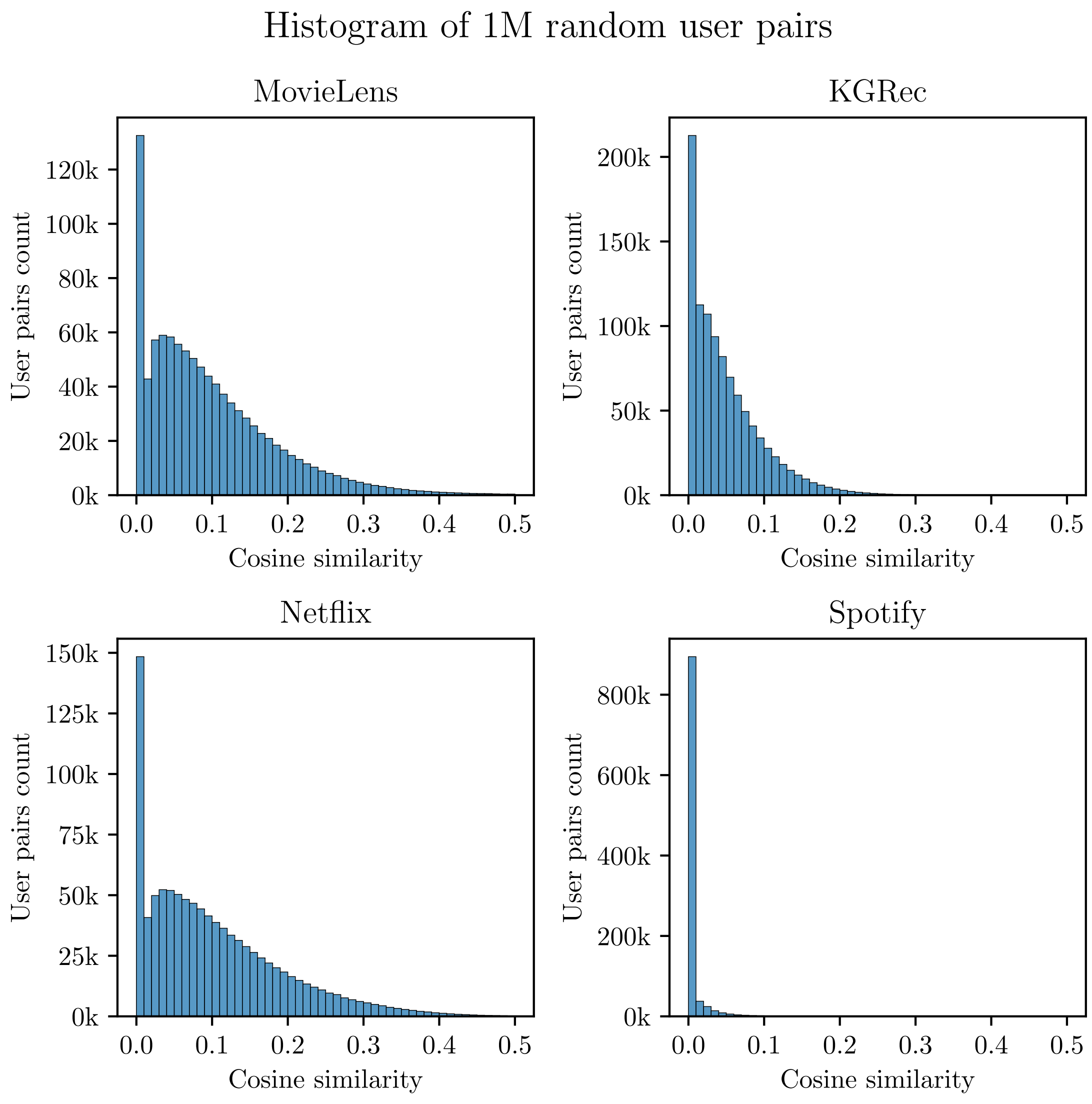


Figure 5.4: Histogram of cosine similarity of 1 million random user pairs from Movie Lens dataset.

Due to the exponentially decreasing number of users with higher similarity scores as can be seen on Figure 5.3 and in detail on 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 the 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 to compute similarity between all possible pairs. We can get a good estimation by sampling some amount of 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 on 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 neighbourhood. For Movie Lens 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. Most important portion of samples between 0.4 to 0.8 where most of the members with 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 neighbourhood of size NS to be:

$$expectedSampleRatioOnInterval(x, N, \lambda) = cdf(exp(\lambda), x+N) - cdf(exp(\lambda), x), \tag{5.4}$$

where cdf stands for the cumulative distribution function of in this case 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.

But 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 uninterested in the actual value, therefore approach using a probability density function of the modeled distribution is better. We can calculate the ration as follows:

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

where pdf represents the probability density function.

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

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

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

The parameter D in scale calculation has a big impact on the scaling, and unfortunately it needs to be tuned to each dataset separately due to the different ratings distributions as can be seen on 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 interquartile range of each dataset.

This gives us a framework as how to assign 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 full 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. And discuss the parameters of the generation process.

Algorithm 1 Generate groups with probability respecting similarity

```
 1: Select scaling parameter $M$ and the desired amount of samples $S$ and amount
    of candidates $C$
 2: for Desired number of samples $S$ do
 3:     Select random pair for users $u1$ and $u2$ from the user-pool
 4:     Calculate similarity of $u1$ and $u2$ and store it
 5: end for
 6: Fit an exponential distribution to the stored values
 7: Calculate the width between 25th and 75th quartile as parameter D
 8: Get its parameter $\lambda$
 9: for Number of desired groups do
10:     Select 1 user $uCaptain$ from the user-pool as main user
11:     Select $C$ random users as $uCandidates$ from the user-pool
12:     for Each user $u$ from $uCandidates$ do
13:         Calculate similarity between $uCaptain$ and $u$ as $sim$
14:         Calculate random candidate weight with (Eq. 5.8):
15:         $weight$ = scale($sim$, D, M) / expectedSampleRatio($sim$, $\lambda$)
16:     end for
17:     for  Desired group size - 1 do
18:         Perform weighted random selection on the candidates using the weights
19:         Add the selected user to the group and remove it from the candidates
20:     end for
21: 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 amount of candidates.

And lastly, the our PRS method. It has 3 parameters in total, where λ is parameter of the exponential distribution fitted to a sample of user similarities. D is the distance and M 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 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, 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 on 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. Further more, we see that PRS generates groups that are comparably or more varied in the similarity as top-k, this is important. Other methods, which manage to create groups that utilize some sort of 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. The reason for this is that most users (playlists) did rate not more than 30 items. Majority ratings are between 0 and 0.1, we therefore need to set the scaling width to a lower value in order 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 amount 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 how reliably generate groups of varying inter-group similarity which are more heterogenous and therefore closer to reality than selecting group members directly based on the desired similarity.

5.4 Dataset tool

We have created a Python CLI tool that allows users to download and process datasets mentioned in 5.1 and create artificial groups with methods described in 5.3.

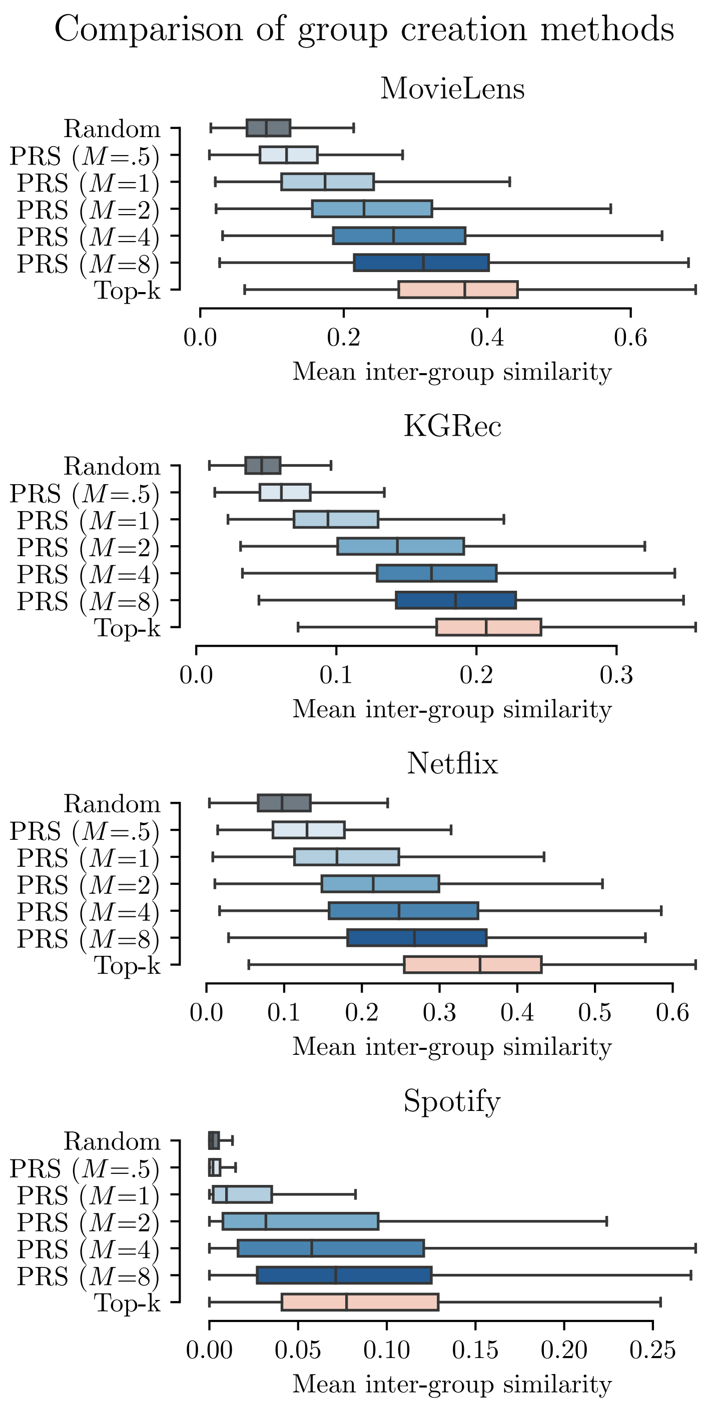


Figure 5.5: Comparison of Random, Top-k and multiple variants of PRS group generation for groups of size 5. Number of candidates for PRS and Top-k was 1000.

add parameters, screenshot and requrements

6. Proposed group recommender system

As we have discussed so far, group recommendation is not an easy task. There is 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 optimizes fairness and relevance.

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 to be fair. But, that is different for tasks where we recommend either a list of items or we have a multiple following recommendation sessions. Then we can tweak our following recommendations in such a way that 'fixes' 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 of consecutively recommended items. All before mentioned methods from 3.3.1 fall into this category. In other words, items are not recommended independently, but a 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 in order to balance fairness among the group members. Most commonly the utility function is a mean of predicted relevance for each user.

And 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 don't 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 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 relevance of items with the 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 [56], the main inspiration for our work are election algorithms that are used for mandates allocation.

Mandates allocation is an important problem that can affect the final distribution of awarded mandates from the underlying vote distribution. The important part of the problem is, what to do on the borders of mandate allocation, 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) 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 [57].

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

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

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

D'Hondt's method minimizes the largest mandate-to-vote ratio as stated in [57], in other words, it tries to minimize unfairness by trying to award mandates in such a way that there are ideally no parties that would on average receive more mandates per each vote. In this sense, it tries to balance fairness. This fact sparked an idea that we maybe also use it to balance fairness in group recommender scenarios.

Unfortunately, direct use of DA is not possible because it does not handle a situation where a single item has utility to multiple users. For a situation with equal user weights and therefore equal votes in the DA terminology, the algorithm would behave the same as FAI.

In order to compensate for this, FuzzDA presented in [58], 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.

6.2 EP-FuzzDA

Let us now describe our algorithm of **E**xactly **P**roportional **FuzzDA**. It is a greedy algorithm that iteratively select 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's preferences. Both these situations are very common in the domain of group recommender systems. We can most certainly find a set of items that closely balance the relevance between all group members. But it is possible, that each of these items do have an item that is better fit for the group over all, but would lead to more imbalance. This could easily lead to recommending mediocre items only for the sake of balancing the fairness.

We therefore want to maximize the balance of items utility for each group member but without hindering the possible additional provided utility to members that overlap in preference with other members or are in other ways 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 item's relevance.

Lets 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 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 add the potential of the candidate item c $TOT(\mathcal{G}, \mathcal{L} \cup c)$ as TOT_c. This represents the expected total plus prospected utility. We then calculate the maximal allowed relevance with the appropriate weight of a user w_u as:

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

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.

On our example in Table 6.1 we have at step 1 $TOT_{c1} = 0.9 + 0.8 + 0.0 = 1.7$ which scaled by uniform weight of 1/3 to $1.7 * (1/3) = 0.57$. Therefore our utility cap for all users (we are selecting a first user, therefore all users started with utility 0) and for candidate item c_1 is set to 0.57. We then calculate the unfulfilled relevance as the total received utility (from already selected items in $\mathcal{L}$) subtracted from maximum allowed maximum utility,

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

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

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 correctly calculate the unfulfilled relevance.

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 $u1$ 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 capped from the bottom to zero. 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 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 EP-rel-sum criterion as

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

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

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

```
1: **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 $w_u$; $\sum w_u = 1$
2: **Output:** ordered list of group recommendations $\mathcal{L}_G$ of size $k$
3: $\mathcal{L}_G = []$
4: $TOT = 0$
5: $\forall u : r_u = 0$
6: **for** 1 to $k$ **do**
7:     **for** $i \in \mathcal{I} \setminus \mathcal{L}_G$ **do**
8:         $TOT_c = TOT + \sum_{\forall u} r_{u,i}$
9:         $\forall u : allowed\_utility_u = TOT_i * w_u$
10:        $\forall u : unfulfilled\_relevance_u = max(0, allowed\_utility_u - r_u)$
11:        $gain_i = \sum_{\forall u} min(r_{u,i}, unfulfilled\_relevance_u)$
12:    **end for**
13:    $i_{best} = \text{argmax}_{\forall i}(gain_i)$
14:    append $c_{best}$ to $\mathcal{L}_G$
15:    $\forall u : r_u = r_u + r_{u,i_{best}}$
16:    $TOT = \sum_{\forall u} r_u$
17: **end for**
18: **return** $\mathcal{L}_G$
```

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

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

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 to optimizing fairness.

One additional nice property that is not present in other group recommendation algorithm 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.

7. Experiments

7.1 Our work

7.2 Proceedings

8. Conclusion

Future work

Check following:
- Introduction with clearly defined goals
- list of literature
- description of the problem that we are solving
- method how will we solve it, precise design, and science
- application of the aforementioned method
- results
- conclusion with what we found out and what are the benefits of my work

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 Serbos, Shuyao Qi, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsaparas. Fairness in package-to-group recommendations. In *Proceedings of the 26th International Conference on World Wide Web*, pages 371–379, 2017.

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

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

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

[56] 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.

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

[58] Ladislav Peška and Štepán 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.

Seznam použitých zkratek

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