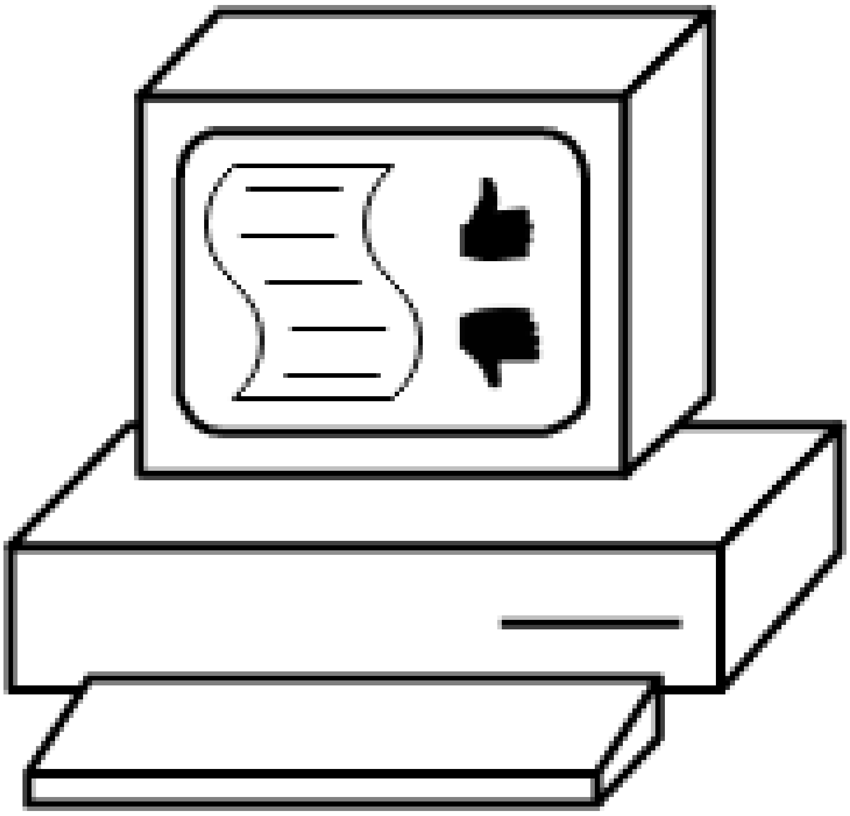
Group Recommendation Using Voting as Mediator



mi908e16
Claus Nygaard Madsen
Lasse Drustrup Christensen
Lukas Nic Dalgaard

AALBORG UNIVERSITY
STUDENT REPORT

© mi908e16, Aalborg University, autumn semester 2016.

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Ø
http://www.cs.aau.dk

Title:
Group Recommendation Using Voting as Mediator

Theme:
Machine Intelligence, Recommendation

Project period:
Autumn semester 2016

Project group:
mi908e16

Participants:
Claus Nygaard Madsen
Lasse Drustrup Christensen
Lukas Nic Dalgaard

Supervisor:
Peter Dolog

Copies: 3

Pages: 46

Date of completion:
December 21, 2016

Abstract:

Group recommendation, while plagued by the inability to obtain an optimal satisfaction for all members, is still improvable. In this report group recommendation is explored with the help of techniques from the information retrieval area and group decision making methods aside from the classic aggregations. From information retrieval we find the methods of rank aggregation and normalized discounted cumulative gain(nDCG). nDCG is used as a satisfaction measure for both the individual group members and the group as a whole. From voting systems the method of single transferable vote is extracted as a decision strategy. These approaches are combined to form what we have dubbed the Borda Transferable Count, a method that our preliminary tests show to give a group about 12% more satisfaction than the classic rank aggregation method of Borda Count.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the authors.

Preface

Before reading this report, it is advised that the reader has a basic understanding of what a recommender system is. Furthermore, understanding content based and collaborative filtering based recommendations is helpful, but not necessary for the sake of this report, however several of the articles referenced to throughout the report assume that the reader knows these concepts. If the reader wishes to study these concepts, the authors recommend reading the relevant sections from "Recommender Systems Handbook"[22].

Throughout the report the authors will refer to themselves through the use of the words, we and our, for the sake of convenience. Additionally, some words and concepts will be abbreviated throughout the report, in these cases the first time the abbreviation is made, it will be presented in parenthesizes, for instance matrix factorization(MF).

Contents

1 Introduction

Recommender systems are a natural response to the problem of information overload.

With a lot of information available a task does not necessarily get easier. When people need to make a choice when faced with complex or abundant information, they simplify it in order to better make sense of the information presented and hasten decision making at the cost of not using all the information available. Recommender systems are a tool to solve that same common challenge for various domains. In philosophy, Thomas Hobbes and others argued that a social contract exists, where people are willing to submit to the will of an outer authority out of self-interest and rationality, whether that authority is a dictator or elected assembly[1]. They both serve the same purpose of making a choice on their behalf.

Today, machine learning helps making informed choices sometimes on behalf of humans for their convenience, trading stocks or working various machine learning tasks. Humans are willingly trading decision power in return for convenience. For a group of people putting aside their individual preferences in order to choose one among several items exists an unspoken social contract between them to accept the result of the process used to reach consensus. A group recommender could be the core of this process, as it could reach a decision in the time it takes humans to pose the question among various other benefits. This motivates the need for a good recommender system.

Group recommender systems change the game and introduce new problems in the area of agreement. To avoid information overload, a group can defer to an outside authority to ease decision making, however doing so transfers some of the problems the group had in agreeing to the system. For a single user, the best choice in a recommendation is what will satisfy the user. However, add more people and disagreement often follows, as some people's preferences are neglected. Kenneth J. Arrow proved this point back in 1948, that no aggregation method could possibly ensure everyone was satisfied[1].

This results in that group recommendations premiere problem being reconciling the differing opinions in the group in a manner that is satisfactory for the group.

1.1 Related Work

There are several different approaches for making recommendations for groups. Two of the most common approaches are to aggregate either users into one superuser or the recommendations for individual users into one super recommendation[23][3][6]. Some of the most conventional aggregation methods used are average, multiplicative, and least misery. However, Delic et al argue that user ratings are not fixed results as they observed how users were accepting of recommendations that were not in line with their

[1]For the Social Contract theory, we refer to http://www.iep.utm.edu/soc-cont/ and https://en.wikipedia.org/wiki/Social_contract

preferences and that those aggregation strategies are not sufficient enough to reflect the result from a decision making process[7].

Carvalho et al looked at Nash Equilibrium as a potential solution to the satisfaction problem[4]. The theory of Nash Equilibrium is largely for a user to vote on what it think other users would prefer among its preferences. Their results showed that Nash Equilibrium performed slightly worse than the Average aggregation strategy but that increases in group size reduced this gap.

The last approach we have looked at is making the group recommendations using voting strategies. Popescu writes about the role of voting in group recommender systems and discusses several voting methods and their characteristics. The idea of using voting to find consensus in a group is not so used within the recommendation domain but is a widely used approach in other domains such as politics[20]. Baltrunas et al use voting to aggregate ranked lists, in this case the items in the ranked list are the candidates and it is then a question of finding those with most votes[2].

These works serve as an inspiration throughout the project.

1.2 Problem Areas

Within the group recommendation field, there are a few areas where there will typically be problems. Some of the problems carry over from recommender systems for individuals, for instance the cold-start problem and the popularity bias. Others are more specific to group recommending, for instance how to replicate group decision making and the satisfaction, while one is caused by the need for automatic verification of the recommending results.

1.2.1 Cold-Start

The cold-start problem is well known within the field of recommender systems, and it is also present in group recommendation[23]. Cold-start refers to the issue that a new user in most cases does not have enough data for a recommendation to be based upon. In group recommenders the problem can be said to be less prevalent as a new user joining a group can, from the systems perspective, act as a passive member of the group until enough data has been gathered. However, there will still be problems when a large number of users in a group are new, as then the system does not work for them. It is a common problem and solutions to it that are applicable to recommender systems in general will likely be usable in group recommendation as well.

1.2.2 Popularity Bias

A common problem found in many recommender systems is the bias towards popular items[9]. Whether this is a problem or not depends on the goal of the recommender, for instance if the goal is to recommend music new to an individual, it might not be the best choice to recommend tracks from the current top 100 hit-list as they have likely already heard them. A common goal for a recommender system is to present items to the user that they might not have known about or thought of on their own, with the same items having a high probability of being liked by the user. In group recommendation the goal is not necessarily to find new items, but rather to find something the group as a whole can be satisfied with. We will follow this belief and disregard the popularity bias.

1.2.3 Group Decisions and Satisfaction

How to properly replicate group decision making is a widely discussed topic within the group recommendation field. This is because the common method of using different aggregation strategies does not always give the best group satisfaction[7]. To combat this, new approaches to simulate the decision making are being continuously developed, along with research into how satisfaction of a group should be measured. It is important to note that the satisfaction of a group does not necessarily reflect the sum of satisfaction of the individual members, as for instance one member might not want to hear specific music on their own but when with a certain group, they might be okay with hearing it anyways. As an example John might not be rating country music highly but when he is with his girlfriend Jane, he listens to it because he knows it satisfies Jane.

1.3 Problem Statement

The purpose of recommending differs depending on the type of domain. Looking at recommendation for individuals, the main purpose of the recommendation is to recommend new items the individual would like while at the same time not just recommending the currently most popular items. This purpose changes if we want to recommend to groups, a group by our definition being 4 people or more. Instead of finding new items, the focus is on satisfying every person in the group to some extent, which does not necessarily require avoiding the popularity bias.

This raises the question of how to measure the satisfaction level in a group of people based on the satisfaction of each persons opinion of the music played. This leads us to the question: *How can recommendations be made to a group of people by reflecting a groups decision making process, ensuring a high level of satisfaction in the group?* This statement opens for questions like:

- How to make recommendations to a group of people based on the users' individual preferences?
- How to measure the level of satisfaction in a group in regards to the items recommended?
- How to reflect group decision making algorithmically while securing satisfaction for the individuals as well as the group as a whole?

1.4 Scenario

For the purposes of this project, the scenario we will be working on is defined as selecting the music to be played in a public area or online gathering. The group size will be 4 or more people. The composition of the group will reflect many small cliques with few connections between communities, as each group member might be acquainted with some other member, but unlikely to know everyone, as the members of the group are seen as random. We do not concern ourselves with the nationality, or background of each group member, as while the group members are spatially bound to the same location when meeting in a public place, this is not the case for an online meeting. With virtual reality becoming more prominent in recent years, a gathering could easily consist of people from all over the world meeting up online in a virtual gathering. This makes it interesting to present a general idea of finding the best music for an impromptu gathering of random individuals.

The group recommender is necessary in this scenario, because sorting through the preferences of even 4 people is an exhausting task, worse than the original motivating factor of information overload for a single person by magnitudes. By removing the people from the decision making process and automating it, not only are the group members free to do other things, but the choice of recommendation does not rely on the initiative of the group, which would lead to no selection being made or a few individuals taking charge in a dictatorial fashion.

2 Background

This chapter describes the theories and knowledge that is used throughout the project. Starting with learning models, with focus on the matrix factorization(MF) method of singular value decomposition(SVD) and variations hereof. Then we describe elements of data preprocessing, testing, and evaluation methods, before the substitute for group decision making by aggregation is described and analyzed. As part of the aggregations we also look at the voting system of single transferable vote(STV).

2.1 Learning Model

While there are several different learning models, we have chosen to focus on a few of them, which are SVD, SVD++, and non-negative matrix factorization(NMF). Specifically we have chosen to look at these because of their offline training capabilities, which are helpful when we have to test multiple different aggregations at a later stage.

2.1.1 Matrix Factorization Models

MF methods are dimensionality reduction techniques used for recommendation that generally find latent features for items, and the propensity for users towards each latent feature. This helps with finding nontrivial correlations in the data as a latent feature could be something that is not at all obvious, but is really important.

A sparse dataset, such as a rating matrix with many items and users, with most cells being zero, we face the *curse of dimensionality*. It is harder to find similar users, because the many dimensions makes differences bigger. Consider a user who has 100 ratings on movies as opposed to 5 ratings on genres. In machine learning, dimensionality reduction concerns itself with mapping these original 100 ratings to a lower dimensional space, such as that of ratings of genres, although these new dimensions are not explicitly known. These are the aforementioned latent features, that now describe each item, rather than the explicitly known features of the original high dimension dataset[24].

MF makes an approximation of a matrix of ratings by decomposing it into multiple matrices. Since MF approximates matrices in an offline step before recommendation, it scales better compared to contemporary collective filtering techniques during recommendation, as most of the calculation work has been done offline. It performs well even when data sparsity is a concern due to the matrix approximation and the dimensionality reduction. As data sparsity grows in a set, the less accurate every method gets. Factorization methods counteract this by compressing the data, making it less sparse, and easier to cluster. As an MF matrix is traditionally updated offline resulting in unchanging matrices, the matrix can be reused as required. This is a benefit when testing various aggregations or configurations of users as most of the calculations are already done as part of the training of the matrix. As the part can be reused for all the tests, it can save some time in practice.

Singular Value Decomposition

On its own, SVD is a dimensionality reduction technique. However, in 2006 Simon Funk popularized it as a recommender method with some modifications[1][12].

Given a matrix of user ratings, SVD works by decomposing the matrix as seen in Equation 2.1 into two matrices called the left and right singular vectors, U and V respectively, and a matrix holding the nonzero singular values on its diagonal, Σ, as per the SVD theorem[10]. For this example, the left singular vector would hold the users on one side, and their inclination towards each latent feature. The right vector would conversely hold all the rated items, and their inclination towards each latent feature[34][24].

$$R = U \times \Sigma \times V^T \tag{2.1}$$

SVD considers as many features as there are ranks, but not all of them are influential. Reducing the number of features to consider is trivial, as the largest singular values stored in Σ have the most influence and are sorted by size.

At this point, it is possible to find some interesting similarities among the users, however to use SVD for recommendation, we must account for the missing ratings. By slightly tweaking the equation, as seen in Equation 2.2, we get an A and B matrix, holding users and items respectively. This results in three matrices looking like Figure 2.1, where F is the number of latent features, throughout the report F and f will be used interchangeably.

$$U \times \Sigma \times V^T = U\Sigma^{1/2}\Sigma^{1/2}V^T = AB \tag{2.2}$$

The A and B matrices can give us our rating matrix, as can be seen in Equation 2.3, before the training starts. After the A and B matrices have been trained, which will be described in next section, this is how the prediction matrix $\hat{R}$ is found.

$$R = AB \tag{2.3}$$

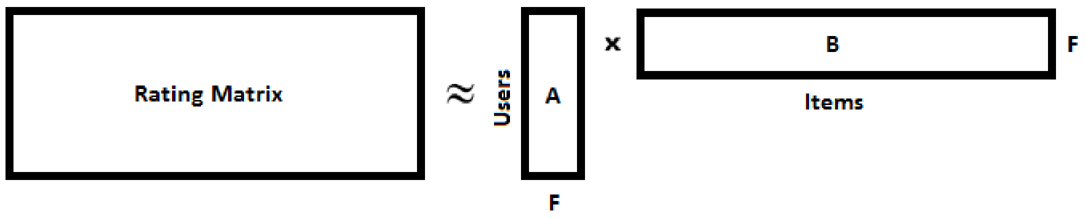


Figure 2.1: The decomposition of a rating matrix using SVD[24]

[1]Source: http://sifter.org/ simon/journal/20061211.html

Moving to the dense subspace of each rated item and user being defined by their features rather than their explicit ratings, data sparsity becomes less of a problem, as missing ratings are squeezed out, as they are subsumed into the lower dimensions for the items.

Equation 2.4 is an example of a decomposition of a 3-by-3 matrix into 3 matrices as per Equation 2.1, the result being the left singular vector, eigenvalues, and right singular vector respectively. Note that the eigenvalues are sorted such that each eigenvalue is smaller.

$$\begin{bmatrix} 1 & 0 & 4 \\ 0 & 2 & 5 \\ 3 & 0 & 6 \end{bmatrix} = \begin{bmatrix} -0.44 & 0.08 & -0.89 \\ -0.54 & -0.82 & 0.2 \\ -0.71 & 0.57 & 0.41 \end{bmatrix} \times \begin{bmatrix} 9.2 & 0 & 0 \\ 0 & 2.45 & 0 \\ 0 & 0 & 0.53 \end{bmatrix} \times \begin{bmatrix} -0.28 & 0.73 & 0.62 \\ -0.12 & -0.67 & 0.74 \\ -0.95 & -0.13 & -0.27 \end{bmatrix} \tag{2.4}$$

In Equation 2.5, the decomposition is split into an A and B matrix as seen in Equation 2.2. Note that the square root of the eigenvalues matrix appears in both calculations.

$$\begin{aligned} A &= \begin{bmatrix} -0.44 & 0.08 & -0.89 \\ -0.54 & -0.82 & 0.2 \\ -0.71 & 0.57 & 0.41 \end{bmatrix} \times \begin{bmatrix} 9.2 & 0 & 0 \\ 0 & 2.45 & 0 \\ 0 & 0 & 0.53 \end{bmatrix}^{\frac{1}{2}} = \begin{bmatrix} -1.35 & 0.13 & -0.65 \\ -1.65 & -1.28 & 0.14 \\ -2.16 & 0.9 & 0.3 \end{bmatrix} \\ B &= \begin{bmatrix} 9.2 & 0 & 0 \\ 0 & 2.45 & 0 \\ 0 & 0 & 0.53 \end{bmatrix}^{\frac{1}{2}} \times \begin{bmatrix} -0.28 & 0.73 & 0.62 \\ -0.12 & -0.67 & 0.74 \\ -0.95 & -0.13 & -0.27 \end{bmatrix} = \begin{bmatrix} -0.85 & -0.36 & -2.89 \\ 1.15 & -1.04 & -0.21 \\ 0.45 & 0.54 & -0.2 \end{bmatrix} \end{aligned} \tag{2.5}$$

It is then possible to reconstruct the rating matrix R and in time, after the training, the prediction matrix $\hat{R}$, using the approach shown in Equation 2.3. An example of this can be seen in Equation 2.6.

$$R = \begin{bmatrix} -1.35 & 0.13 & -0.65 \\ -1.65 & -1.28 & 0.14 \\ -2.16 & 0.9 & 0.3 \end{bmatrix} \times \begin{bmatrix} -0.85 & -0.36 & -2.89 \\ 1.15 & -1.04 & -0.21 \\ 0.45 & 0.54 & -0.2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 2 & 5 \\ 3 & 0 & 6 \end{bmatrix} \tag{2.6}$$

Gradient Descent

In Funk's blog, the gradient descent optimization algorithm is proposed as a common method to learn the A and B matrices to minimize the error, as found using Equation 2.7. The error is calculated as the euclidean distance between our known ratings, R, and our predicted ratings $\hat{R}$[17].

$$\text{Error} = |R - \hat{R}| = |R - AB| = \sum_{u=1}^{U}\sum_{i=1}^{I}\left(R_{ui} - \sum_{f=1}^{F} A_{uf}B_{fi}\right)^2 \tag{2.7}$$

Given as a function, the gradient will point in the direction of the largest error increase, as shown in Equation 2.8. So to minimize the error, steps are taken towards the negative of the gradient[18].

$$\begin{aligned} \frac{dError}{dA} &= -(R - AB)B^T = \sum_{i=1}^{I}(R_{ui} - \sum_{n=1}^{N} A_{un}B_{ni})B_{fi} \\ \frac{dError}{dB} &= -(R - AB)A^T = \sum_{u=1}^{U}(R_{ui} - \sum_{n=1}^{N} A_{un}B_{ni})A_{uf} \end{aligned} \tag{2.8}$$

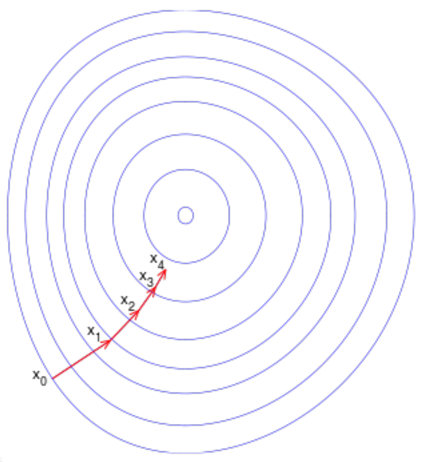


Figure 2.2: Gradient descent showing steps as it converges toward a local minimum[2]

Taking the average error when finding the derivative is a costly operation for learning the A and B matrices, as it sums up the error for every user-item pair across all of their features. This can be expressed by the running time $\Theta(U \times I \times F)$.

The common solution is using the stochastic gradient descent. For this, a single observed rating is selected at random for each update step of the gradient descent, rather than summing up over all the item-user pairs to find the exact gradient. The updates will be a bit more erratic than seen in Figure 2.2[2], but will overall converge towards a local minimum.

The learning rate describes the speed at which the error is corrected at each step. A high learning rate results in fewer steps and a lower chance of getting stuck in a local minimum, however a high learning rate makes convergence more difficult as the model can end up dancing around the minimum by taking too large steps near the end. A too small learning rate can result in the model settling in a local minimum for the cost function, unable to escape, as the gradient of the error will always point back towards the nearest local minimum and is slow to train. A learning rate is usually between 0.1 and 0.001, but it is recommended to take an exploratory approach. In Equation 2.9, η is the learning rate.

$$\begin{aligned} A_{uf} &\leftarrow A_{uf} + \eta(R_{ui} - \sum_{n=1}^{N} A_{un}B_{ni})B_{fi} \\ B_{fi} &\leftarrow B_{fi} + \eta(R_{ui} - \sum_{n=1}^{N} A_{un}B_{ni})A_{uf} \end{aligned} \tag{2.9}$$

Regularization helps avoid overfitting, where a model fits its training data too closely

[2]Source: https://en.wikipedia.org/wiki/Gradient_descent

so that it predicts extremely well but only within the training data. It is added as an additional modifier when updating the model as can be seen in Equation 2.10, where λ is the regularization value. It makes the resulting model more general, which improves accuracy when moving beyond the training set.

$$\begin{aligned} A_{uf} &\leftarrow A_{uf} + \eta(R_{ui} - \sum_{n=1}^{N} A_{un}B_{ni})B_{fi} - \lambda A_{uf} \\ B_{fi} &\leftarrow B_{fi} + \eta(R_{ui} - \sum_{n=1}^{N} A_{un}B_{ni})A_{uf} - \lambda B_{fi} \end{aligned} \tag{2.10}$$

After the A and B matrices have been trained it is possible to construct a prediction

SVD++

As the name implies, SVD++ is an extension on SVD. Where SVD in general only works with explicit data, SVD++ takes into account the implicitly gathered data as well. Take for instance a user that has a number of items they have shown some indirect interest in, for instance hovering over the items to see some information about them, SVD++ would specifically account for this set of potentially interesting items[25].

The standard model for SVD can be seen in Equation 2.3. This could be modified in order to only find one rating, which would look like Equation 2.11.

$$\hat{R}_{ui} = \sum_{f=1}^{f}(A_{uf}B_{fi}) \tag{2.11}$$

In SVD++ the last part of the model is altered, to reflect how users are characterized based on the items they have rated, as seen in Equation 2.12. Here $R(u)$ is the set of items rated by the user u, and y_j is a feature vector. A_u is a free user-feature vector of size f, equal to the number of latent features.

$$\hat{R}_{ui} = \sum_{f=1}^{f}\left(B_{fi}\left(A_{uf} + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j\right)\right) \tag{2.12}$$

There is in theory no limit to the number of itemsets that can be inferred and accounted for, as each would simply have its own item feature vector, as seen in Equation 2.13.

$$\hat{R}_{ui} = \sum_{f=1}^{f}\left(B_{fi}\left(A_{uf} + \left|N^1(u)\right|^{-\frac{1}{2}} \sum_{j \in N^1(u)} y_j^{(1)} + \left|N^2(u)\right|^{-\frac{1}{2}} \sum_{j \in N^2(u)} y_j^{(2)}\right)\right) \tag{2.13}$$

SVD++ in general performs better than SVD, however if there is no implicit data, the advantage in using SVD++ becomes negligible.

Non-negative Matrix Factorization

NMF is a constraint on the standard MF. All values are equal to or higher than 0, and when learning the model, no values can go below 0, which is ensured by setting any values that have become negative back to 0 before the next iteration. This is helpful for speeding up the learning in certain domains, for instance finding objects or parts hereof in images[14].

As shown in Equation 2.14, given a rating matrix of size *m-by-n*, for example m users and n items, we can find non-negative matrix factors W and H of size *m-by-f* and *f-by-n* respectively, such that they approximate V. During this f is usually chosen to be smaller than the rank of the m by n matrix, as to reduce the size of W and H compared to the original matrix, V[13].

$$V \approx WH \tag{2.14}$$

As an example of NMF, if we consider the A and B matrices found in Equation 2.5 to be our potential W and H matrices without the constraint, we would with NMF get a result as in Equation 2.15.

$$\begin{aligned} W &= \begin{bmatrix} 0 & 0.13 & 0 \\ 0 & 0 & 0.14 \\ 0 & 0.9 & 0.3 \end{bmatrix} \\ H &= \begin{bmatrix} 0 & 0 & 0 \\ 1.15 & 0 & 0 \\ 0.45 & 0.54 & 0 \end{bmatrix} \end{aligned} \tag{2.15}$$

Choice of Recommender System

For this problem, we went with MF. We selected MF because the offline training makes us able to reuse a model once it has been trained for the multiple tests. This speeds up testing of various methods using the model, as a small change does not require retraining the entire underlying model. SVD also has several other notable benefits, such as finding otherwise hidden correlations in the data through dimensionality reduction.

Among the different MF approaches, we went with the standard SVD, because SVD is a standard method, so it is more easily replicated by other studies and there are many libraries available for it. The precision of the recommender is also not the main focus of this project, as the satisfaction of the group is not improved by a better precision.

2.2 Data Preprocessing

When working with data and datasets, it is important to ensure that the data format is structured in a way that is usable by the algorithms being implemented or tested[24].

To this end data preprocessing is an obvious part of the solution. Typically this consists of formatting, filtering, and splitting the data.

As the data is not always in the correct format, it is sometimes necessary to perform some sort of formatting on it, for instance transforming a tab-separated file into a proper matrix-format for use in matrix related calculations.

Filtering or cleaning is simply the art of cleaning the dataset of irrelevant data, such as the age and gender of users in an experiment, where who and what the users are play no role. To some extent it is also used to correct data entries that are corrupted or otherwise unusable, for instance uninterpretable entries being replaced by either what was supposed to be there or a non-influential value.

2.3 Testing and Evaluation

This section covers the testing and evaluation methods we will use. In specific k-fold testing and our evaluation metrics will be explained.

2.3.1 K-fold Testing

Stemming from cross validation in the field of statistics, k-fold testing is a commonly used technique in regards to recommender systems[26]. The basic principle in the method is to split the dataset into *k* equally sized parts or folds, and then use one of the folds as the testing or validation set, while the rest of the folds are used as training data. This process is then repeated *k* times, so each fold is used as the validation set once. After it has been repeated *k* times, the results can then be averaged to find an overall estimation of the error in the model. An example of how the k-fold ordering could be structured can be seen in Table 2.1. Typically, either 5-fold or 10-fold testing is what is used.

	Training			Validation
First	1	2	3	4
Second	2	3	4	1
Third	3	4	1	2
Forth	4	1	2	3

Table 2.1: Example of the rotation of folds in a 4-fold dataset

2.3.2 Evaluation Metrics

Here we will cover the various evaluation metrics used in the project. MAE and RMSE are considered for the recommender for tuning, and nDCG is presented for evaluating satisfaction with a recommendation.

Root Mean Square Error

The root mean square error(RMSE) is a common accuracy measure for recommender systems[27]. For the algorithm, RMSE is calculated using the Equation 2.16. τ is the set of user item pairs used to validate the system, with u and i being user and item, respectively. $\hat{r}_{ui}$ are the predicted ratings for a user item pair, from which we subtract the actual rating, r_{ui}, to find the error.

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\tau|} \sum_{(u,i)\in\tau} (\hat{r}_{ui} - r_{ui})^2} \tag{2.16}$$

An alternative, mean absolute error(MAE), as seen in Equation 2.17, takes the euclidean distance rather than the squared difference[27]. As opposed to MAE, RMSE penalizes larger errors relatively more harshly.

$$\mathrm{MAE} = \frac{1}{|\tau|} \sum_{(u,i)\in\tau} |\hat{r}_{ui} - r_{ui}| \tag{2.17}$$

Normalized Discounted Cumulative Gain

Discounted cumulative gain is used to measure the quality of a ranking through the relevance of the result set. It is most commonly used in the information retrieval field for determining the quality of the result of a query, however it is also useful for getting a measure of satisfaction for an ordering of recommended items. That is because recommendation and the results of a search query are both solutions to the problem of information overload that presents a set of possible items[28][2].

For a group recommendation this works by comparing the group recommendation with the individual preferences. In this way, the relevance of an item is translatable to a ranked list of recommendations, as a recommendation can be viewed as a fixed query.

The measure grew out from the Cumulative Gain measure, which takes the relevance score of each result in a set without considering the position in the set. Equation 2.18 shows the process. p denotes the number of positions in the ranking.

$$\mathrm{CG}_p = \sum_{i=1}^{p} rel_i \tag{2.18}$$

The addition of the discounted property adds a consideration over the ranking. For most lists, it is assumed that a user will consider each item in order from first to last, making the most important position the first item. So it follows that the most relevant item should be the first item. This justifies penalizing a deviance from ordering by the most relevant item first. The score is calculated using Equation 2.19.

$$\mathrm{DCG}_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} \tag{2.19}$$

Normalized discounted cumulative gain compares the DCG with an ideal ranking as to normalize the score, where a perfect ordering results in a score of 1, so it can be compared for systems using rankings of different lengths. The score is calculated using Equation 2.20. The IDCG is the ideal DCG with basically is the elements in DCG sorted in the order of relevance.

$$\mathrm{nDCG}_p = \frac{\mathrm{DCG}_p}{\mathrm{IDCG}_p} \tag{2.20}$$

As an example of finding the satisfaction of a user based on the recommendations for the group, the rating of a group of users can be as seen in Table 2.2. User U1 will be the focus of this example, and in Table 2.3, the groups 4 most preferred items, based on average, is shown together with user U1's 4 most preferred items. As can be seen, item M8 and M4 overlaps for the two lists giving us a relevance scores for U1 on 9, 9, 0, 0. The two last relevance scores are zero because they are not present in U1's top 4 preferences, which we look at in this case.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
U1	2	4	5	9	1	3	7	9	8	6
U2	5	8	9	4	3	6	2	10	1	7
U3	4	6	2	7	8	3	1	4	10	9
U4	10	8	2	7	1	4	6	5	8	3
U5	1	5	9	5	4	2	8	10	2	7
Total	22	31	27	32	17	18	24	38	29	32

Table 2.2: Example of ratings for a set of items

	I1	I2	I3	I4
Group ranking	M8	M4	M10	M2
U1 ranking	M8	M4	M9	M7

Table 2.3: Indices of the group preferences and user U1

The cumulative gain of the example results in a high count in Equation 2.21, as to be expected as the group and U1 share many preferences in the list. When we consider the positioning in Equation 2.22, there is only a small drop in the score, as the bulk of U1's points are at the top positions where the penalty is smaller.

$$\mathrm{CG}_4 = \sum_{i=1}^{4} rel_i = 9 + 9 + 0 + 0 = 18 \tag{2.21}$$

$$\mathrm{DCG}_4 = \sum_{i=1}^{4} \frac{rel_i}{\log_2(i+1)} = \frac{9}{log_2(2)} + \frac{9}{log_2(3)} + \frac{0}{log_2(4)} + \frac{0}{log_2(5)} = 9 + 5.68 + 0 + 0 = 14.68 \tag{2.22}$$

To normalize the score, we compare it to the ideal ranking, which is U1's own ranking which would be 9, 9, 8, 7. The score for the ideal ordering is calculated in Equation 2.23. The level of satisfaction for U1 for this recommendation is 0.68 equal to a 68% points, as per Equation 2.24.

$$\mathrm{IDCG}_4 = 9 + 5.68 + 4 + 3.05 = 21.73 \tag{2.23}$$

$$\mathrm{nDCG}_4 = \frac{\mathrm{DCG}_4}{\mathrm{IDCG}_4} = \frac{14.68}{21.73} = 0.68 \tag{2.24}$$

This is an above average score, because even though only 50% of U1's list is represented, it is the highest ranked items and they are highly ranked on the groups list, which is why the percentage point is 68% and not 50%.

2.4 Aggregation

When working with a group recommender system, it is hard to avoid using aggregation of some kind, to make group-like decisions. Despite this, recent research has shown that current aggregation methods function poorly as a direct substitute for group decisions[7]. Therefore, we will in this section explore some of the basic aggregation strategies, as well as a few alternate approaches, that could also be considered to make group decisions.

While specific strategies are as numerous as the number of group recommender systems, they are generally derived from a few common strategies[23]. Most of these strategies will here be presented using the ratings in Table 2.4, while the full overview of the results of aggregation can be seen in Table 2.5.

	A	**B**	**E**	**D**	**E**	**F**	**G**
John	9	4	7	10	5	1	2
Jane	10	10	5	1	8	7	4
Bob	1	8	4	4	6	10	6
Hilda	1	7	2	9	5	10	3

Table 2.4: Arbitrary examples of ratings given by four people. The ratings were chosen to showcase the differences in the aggregation strategies.

Plurality Voting attempts to gain satisfaction by selecting the items with the highest ratings for the majority of the group first, and then repeating this strategy removing any items already selected. Outlier cases can result in a minority of highly dissatisfied users, simply because their ratings are insignificant. For instance item A being rather highly scored despite two people hating it.

Average is one of the simplest ways of aggregating, by simply taking the average of the ratings for an item. As with any averaging, this is not necessarily the best approach as low ratings can be overshadowed by a significantly higher amount of high ratings. This is for instance the case with item F, where the low rating from John is overlooked in favor of the other people's ratings.

Multiplicative attempts to improve satisfaction by multiplying each rating for an item with each other, so outlier ratings at either end of the rating scale becomes more significant. This is especially true for the lowest rating, commonly adjusted to be 1, as the rating adds nothing to combined rating. Thereby items receiving the lowest rating by even a small amount of group members quickly fall towards the bottom of the recommendation. For instance this is seen in item D, where Jane's low rating drags the overall score down.

Approval Voting transforms the ratings into points for each item. The points are awarded for each group member who have a rating for the item above a certain threshold. In small groups the strategy leads to many ties, in larger groups these ties become more uncommon. In the example item E is approved by every member, and with the threshold set at 5 this is an indication that no member would be dissatisfied with this item.

Least Misery attempts to reduce the amount of misery felt by the group by using the lowest rating for an item. This leaves the items to be easily be sorted by how well they are liked by the person who likes them the least. However it also has outlier cases with items well liked by the vast majority but disliked by a single person leaving it with a low rating. This is clearly seen in item F, where John's low rating makes the item seem irrelevant despite it being highly regarded by the other three.

Most Pleasure attempts to maximize the satisfaction of the group by only using the highest rating for an item. Like the least misery strategy, this suffers from outlier cases going in the other direction, a single person rating an item disliked by the vast majority leaves the item with a high rating. This can be seen in item A, where it ends up with a tie for best choice, despite being hated by half the group members.

Average Without Misery works on the concept of removing as much misery as possible, this strategy removes items who have any ratings below a certain threshold, before averaging the ratings of the remaining items. However average without misery suffers from outlier cases just as much as least misery and most pleasure. The example showcases another problem, where even if one person has given a rating below the threshold, then the item is completely disregarded, therefore this strategy is unsuitable at even small group sizes. In the example only two out of seven items are left for consideration.

Fairness goes by the principle of everyone-gets-a-choice, is the essence of fairness. The items are selected in turn by each group member, and in the purest form the selection is based purely on that member's terms without regarding ratings from others. However ratings from other group members are typically used in case the selecting member has a tie between two or more items. In the example the pure form is used and notable items include item E, which has fairly high ratings from everyone but not the highest rating from anyone, leaving it to be selected second to last.

Dictatorship is also called most respected decides. Letting one person and their ratings decide everything is an approach often used in real world scenarios, for instance music being decided by a dj or the movie to watch being decided by the host. However deciding which person to use as the “dictator” is typically handled using trust or respect, hence the alternative name for the strategy. Trust is difficult to measure, especially the more randomly put together the group is.

	A	B	E	D	E	F	G
Plurality Voting	5	4	2	6	3	7	1
Average	5.25	7.25	4.5	6	6	7	3.75
Multiplicative	90	2240	280	360	1200	700	144
Approval Voting (Threshold 5)	2	3	2	2	4	3	1
Least Misery	1	4	2	1	5	1	2
Most Pleasure	10	10	7	10	8	10	6
Average Without Misery (Threshold 4)	-	7.25	-	-	6	-	-
Fairness (Start at John)	6	4	3	7	2	5	1
Dictatorship (Hilda)	1	7	2	9	5	10	3

Table 2.5: Examples of the scores that results from applying the individual aggregation strategies to the ratings from Table 2.4. All values have been adjusted so the highest value is the ’best’.

2.4.1 Simulating Group Behavior

While the strategies presented until now forms the basis for many aggregation strategies, part of the scenario we work with implies that more than one item should be found for the group. Masthoff found in a study, that certain of the base strategies better simulated normal group decision making, such as average, average without misery, and least misery[15]. However when the groups were presented with an ordered list of recommendations made from the basic strategies, the groups indicated that multiplicative, average, and most pleasure would bring the most satisfaction to the group. However it should be noted that the groups in the study were measuring the satisfaction of a fictive group, and not their own. That being said, it was found that the ordering of the recommendation had a large impact on the overall satisfaction.

2.4.2 Rank Aggregation

Another aspect of aggregation is rank aggregation, where the input are lists of preferences. The goal as with any aggregation is then to find the best ordering possible. The best ordering is also known as the Kemeny Optimal[8]. Rank aggregation has to the best of our knowledge settled on two main methods, Borda and Footrule, both of which are good approximations of Kemeny optimal aggregation.

Borda or Borda Count(BC), works by assigning a score for each member to each item based on the item's placement on the members ordered list[2][29]. Then the total score for an item is the sum of scores it got from each user. Often the individual scores are inversely proportional to the placement, so on a top n list item 1 would get a score of n, item 2 a score of $n-1$ etc..

Performing BC of two given ranked lists $L_1 = a, b, c, d, e$ and $L_2 = c, a, f, e, d$ would work as follows. Assign scores to each item in each list $S_1 = a = 5, b = 4, c = 3, d = 2, e = 1$ and $S_2 = c = 5, a = 4, f = 3, e = 2, d = 1$. Then sum up the scores for each item: $a = 9, b = 4, c = 8, d = 3, e = 3, f = 3$. According to B the best ordering would then be a, c, b, def, where d, e, and f are interchangeable.

Footrule or Spearman's Footrule, is an aggregation method where the average Spearman footrule distance is minimized on the input rankings[2]. Between two lists, the Spearman footrule distance is the sum of absolute difference of the rank positions of each item between the two lists. If an item is present in one list of top n items but not in another, the item is placed in the list it is missing from at rank $n+1$. For instance take the two ranked lists L_1 and L_2 from earlier. First L_1 is designated as what is called the pattern-vector which has coordinates based on the ranking in the list, e.g. a = 1, b = 2, etc.. As the item f is present in L_2 but is missing in L_1 it is added as f = 6. Subsequently item b is also added to L_2. This results in L_1 being the pattern-vector $(1, 2, 3, 4, 5, 6)$ while L_2 becomes the vector $(3, 1, 6, 5, 4, 2)$. The footrule distance between these are now easily calculated as follows $|3-1| + |1-2| + |6-3| + |5-4| + |4-5| + |2-6| = 12$.

Finding the aggregation then becomes a task of finding the minimum cost maximum matching in a bipartite graph. This should be the complete weighted bipartite graph (C, P, W), where C is the first set of nodes, the items being ranked; P is the second set of nodes, the positions to be filled; and W is the weight given by the scaled footrule distance between item c and position p on the form seen in Equation 2.25, where τ_i are the ranked lists.

$$W(c,p) = \sum_{i=1}^{k} \left| \frac{\tau_i(c)}{|\tau_i|} - \frac{p}{n} \right| \tag{2.25}$$

2.4.3 Voting

A final approach, that could also be regarded as an aggregation, is voting. While there are many different voting and electoral systems, we have decided to focus on only one of these, namely STV, a system which has seen real world application for more than 90 years[16]. STV is speculated to be one of the best electoral systems, in the sense that it seeks to reduce the number of wasted votes. The system operates on ranked lists of candidates, which can directly be translated into a ranked list for a ranked aggregation, making it relevant to take into account for our approach.

The basis of the system is having a number of open positions, also called seats; a number of candidates for the positions; the total number ranked lists with the votes; and a threshold that each candidate have to pass to guarantee their seat. The threshold can be simply calculated as the total votes divided by the number of seats[33].

Following the flow depicted in Figure 2.3, firstly the votes for each candidate are summed up and if any candidates exceed the threshold the excess votes, is transfered to other candidates based on the other candidates in the voting lists, typically based on the proportions, for instance candidate A has reached the threshold and amongst his voters 30% prefers candidate B as their second choice, 60% prefers candidate C, and the remaining have no secondary preference. Therefore, the excess votes should be assigned according to these proportions, while the 10% that had no secondary preference would be discarded which results in a slight loss of votes.

Afterwards, the candidate with the currently lowest amount of votes will be eliminated and their votes partitioned out similarly to before. This is repeated until all the seats have been filled, however because votes are occasionally tossed out, the procedure should also stop when the number of candidates has been reduced to be equal to the number of seats. The method does not inherently handle cases where there are more seats than candidates.

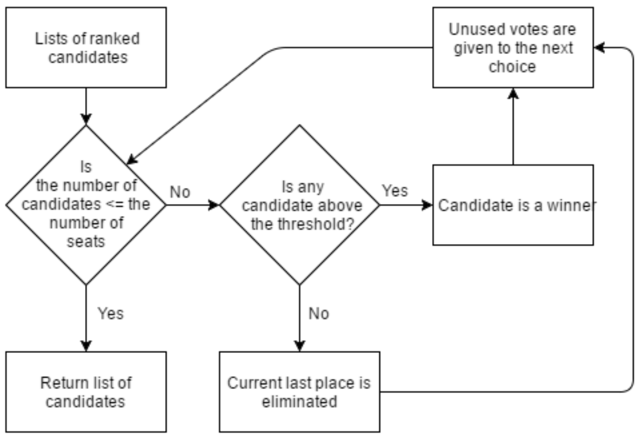

Lists of ranked candidates
Is the number of candidates <= the number of seats
No
Is any candidate above the threshold?
Yes
Candidate is a winner
Unused votes are given to the next choice
Yes
No
Return list of candidates
Current last place is eliminated

Figure 2.3: Flow of an STV voting process

2.4.4 Discussing the Aggregations

In an attempt to satisfy a group as much as possible, it is not enough to look at the current aggregation and voting methods in isolation. Many typical aggregations would in theory perform worse as the group size increase, and voting systems are designed for use in scenarios with many voters and relatively few candidates. While rank aggregations gives credence to a possibly good solution, there is room for improvement. That being said, as we are interested in satisfying not only the group but also the individual members, to the best degree possible, using the base ideas of a voting system, such as STV, makes sense. While STV on its own would likely fail for our scenario, due to the large number of candidates compared to the number of voters, the idea of transferring votes would help maintain every voters importance. This approach is discussed further in Section 3.1.3.

Furthermore, as BC performs similarly to the more complex Spearman's Footrule, when it comes to the satisfaction they deliver, we will be basing our further exploration on BC[2].

3 Aggregation Design

This chapter introduces the, to the best of our knowledge, novel ideas for aggregating ranked lists, Borda Weighted Count(BWC), Borda Escaleted Count(BEC), and Borda Transferable Count(BTC). Each method aims to increase the groups combined satisfaction with a ranked list of items. Additionally, an explanation is given of the baseline aggregations used.

3.1 Novel Aggregation Methods

In this section new aggregation methods are presented. The methods are described and exemplified using the small randomly generated dataset seen in Table 3.1.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
U1	1.2	2.9	3.0	5.0	1.0	1.7	4.1	5.0	4.9	4.0
U2	2.2	3.6	4.3	2.1	1.8	2.8	1.3	5.0	1.1	3.5
U3	1.9	2.7	1.6	3.9	4.3	1.7	1.2	1.9	4.5	4.4
U4	4.4	3.9	1.6	3.7	1.0	2.7	3.6	3.0	3.9	2.6
U5	2.3	3.5	4.6	3.5	3.4	3.0	4.5	4.7	3.0	3.9

Table 3.1: Randomized table of ratings for 5 users with 10 items.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
U1	2	4	5	9	1	3	7	9	8	6
U2	5	8	9	4	3	6	2	10	1	7
U3	4	6	2	7	8	3	1	4	10	9
U4	10	8	2	7	1	4	6	5	8	3
U5	1	5	9	5	4	2	8	10	2	7
Total	22	31	27	32	17	18	24	38	29	32

Table 3.2: Borda Count results from the randomized example

3.1.1 Borda Weighted Count

Using BC as a base and adding a weight to the items based on the number of times they occur in the top-k lists would help increase the score of those items multiple users have rated high enough to get onto their top-k list. Our implementation will use a simplistic weight of adding 1 additional point to the final tally of the scores for each time they appeared in the top-k lists the group members produced. Depending on the size of *k* it could very well be that the weighting factor should be changed as the weight used would have little influence when scores of more than 50 are handed out.

An example of how the scores from the random samples top-4, Table 3.3, have influenced the ordering can be seen in Table 3.4. The order has not changed per say, number 1 and 2 is T8 and T9 respectively just as in BC, however BWC gives a shared third place to T3, T4, and T10, whereas BC gave the third place to T3 and had T4 and T10 on a shared fourth place.

3.1.2 Borda Escalated Count

BEC is based on the idea that BC does not necessarily give enough distinction to the top of the top-k items. This method will increase the scores given by the normal BC by some amount based on where in the top-k list the item appear. Our implementation will split the top-k list into three parts, the upper part will receive an extra 3 points to all items, the items in the middle part will be given 1 extra point, while the lower part will receive no extra points.

The example of how the scores would be are again presented in Table 3.4. Here T10 is no longer amongst the four items scoring the highest, while the others are placed as in BWC.

3.1.3 Borda Transferable Count

As it was determined in Section 2.4.3, STV was unsuitable as a selection process, but by modifying the voting we believe that the problems with using STV in recommender systems can be accounted for. The change proposed for BTC is changing the single vote into a BC score, so instead of a user only having 1 vote on a top-10 list, they would have 55, the highest rated item with 10, second item with 9, etc.. This approach should allow for more reliable selection, as users no longer end up only voting on their own highest rated items. Of course this relies on at least a few overlaps between items on the top-k lists.

Here is a simplified example of how it will perform if it is used on the random sample with top-4, see Table 3.3 and giving the results seen in Table 3.4.

1. Threshold calculated to be 12.5, but for simplicity it is set to 13.
2. No candidate exceeds the threshold, so instead the current worst candidate T5 is eliminated, their votes transferred to their highest priority T9.
3. T7 is now the next to be eliminated, 1 vote moved to T4 and 2 votes to T8, T8 is now locked in.
4. T8 hit our simplified threshold without exceeding it so no votes are transferred further, instead T1 is the next to be eliminated, only one user voted on it but their secondary choice is split between T2 and T9 so 2 votes will be transferred to each.

5. Next up is T10 being eliminated and going by the lists, U2 and U3 would instead give their 4 combined votes to T9, while U5 would give a single vote to T8, but as that candidate is already locked in the vote is instead transferred to T3, the secondary choice.

6. T9 now exceeds the threshold and 3 votes have to be transferred to other choices. For sake of simplicity the votes are handed out for users to redistribute equally, so U1, U3, and U4 each get one vote to reassign, U1 and U3 both give it to T4, while U4 gives it to T2.

7. Next T2 is eliminated and the votes distributed equally between T3 and T4.

8. There are now only four candidates left for four seats, and going by the order they were locked in and then the number of votes they got the order would be T8, T9, T4, and then T3.

The resulting ordering is again not too different from the ones seen before, however it has been definitively reduced to only four candidates. The random data most definitely favored T8 and T9 in these small examples, so the fact that they are solidly chosen as the first and second place is fitting. Third and fourth place has been switched in BTC compared to the other orderings, however they have generally been close in general. That being said, the ordering of the candidates throughout the process have a large impact on which candidates that gets eliminated, which leads to some inherent bias.

	Rank 1	Rank 2	Rank 3	Rank 4
U1	T4 (3)	T8 (3)	T9 (2)	T7 (1)
U2	T8 (4)	T3 (3)	T2 (2)	T10 (1)
U3	T9 (4)	T10 (3)	T5 (2)	T4 (1)
U4	T1 (4)	T2 (2)	T9 (2)	T4 (1)
U5	T8 (4)	T3 (3)	T7 (2)	T10 (1)

Table 3.3: Ranking of the top-4 items for each user from Table 3.1, BC score in parenthesis

	T1	T2	T3	T4	T5	T7	T8	T9	T10
BC	4	4	6	5	2	3	11	8	5
BWC	5	6	8	8	3	5	14	11	8
BEC	7	6	8	8	3	4	18	13	6
BTC	-	-	10.5	11.5	-	-	13*	13*	-

Table 3.4: Scores from each method on top-4, in BTC the * indicates the candidate hitting the threshold

3.2 Control Aggregations

Implementing and testing the aggregation methods described in Section 3.1, would not provide any useful data on its own. Therefore two aggregation methods will be used as a measuring baseline. The first will be BC that will represent a solid comparison baseline, since the new methods being tested are modifications of BC, while the second will be a random selection method. If the new methods can not, on average, outperform a random selection, then there would be no reason to use the method at all.

4 Group Recommendation

In this chapter we will describe the implementation setup. We start by shortly describing the choice of dataset and the preprocessing thereof. Then we move on to the implementation SVD, which arguably also can be categorized as a preprocessing step. Next we describe, how we extract the users ranked lists from the prediction matrix, we get from SVD. Then we show how we have implemented the aggregation algorithms described in the previous chapter. Lastly, we have a small discussion regarding the implementation.

4.1 Data Processing

For this project we were using the MovieLens 100k dataset, which contains 100.000 ratings, between 1 and 5, from 943 users spread out on 1682 movies[11]. We also looked at a dataset from last.fm called 1K[5], but we decided to use the MovieLens set because it came prepackaged with a 5-fold test setup, which saved us some processing work. We do this as it is easier for testing purposes, as it will still be applicable for the scenario given in Section 1.4, if given a proper dataset for the domain to train with.

In order for us to use the data for SVD we needed to present it in a matrix. We constructed 6 different $Users \times Movies$ matrices. One for each of the folds for both training and testing containing the 80000 ratings and 20000 ratings respectively and a matrix with all the ratings.

4.2 Individual Recommendation

In Section 2.1.1 we determined that SVD would be a suitable method of recommendation for this project. We have experimented with two different approaches with slight differences in how they do regularization and in how the A and B matrices are constructed.

The first approach was the Funk-SVD which can be seen in Equation 4.2, where Equation 4.1 defines the error[12]. In his blok Simon Funk[1] mentions that the algortihm uses a method somewhat like Tikhonov regularization[2], denoted as λ in the equation. We have made some adjustments compared to Funks documentation[12]. Instead of calculating the values of matrix A and B based on average movie rating and average offset for users, our initial test showed the best performance when populating them with random variables between 0 and 1.

$$error = R_{ui} - \sum_f A_{uf} B_{fi} \tag{4.1}$$

[1]http://sifter.org/ simon/journal/20061211.html
[2]https://en.wikipedia.org/wiki/Tikhonov_regularization

$$\begin{aligned} A_{uf} &+= \eta * (error * B_{fi} - \lambda A_{uf}) \\ B_{fi} &+= \eta * (error * A_{uf} - \lambda B_{fi}) \end{aligned} \tag{4.2}$$

The second approach can be seen in Equation 4.3, which is also described in Section 2.1.1. As mentioned this approach has some differences compared to Funk-SVD. First of all, it uses a simple weighted-decay for regularization and not Tikhonov. Second, the A and B matrices for this approach is populated using the SVDS method in MatLab, which is similar to Equation 2.2 in Section 2.1.1.

$$\begin{aligned} A_{uf} &+= \eta * error * B_{fi} - \lambda A_{uf} \\ B_{fi} &+= \eta * error * A_{uf} - \lambda B_{fi} \end{aligned} \tag{4.3}$$

In Listing 4.1, it is shown how we trained both of the algorithms. We have a while-loop that runs through a specified number of iterations or as long as the RMSE value does not grow too much. We train both methods using stochastic gradient descent so we select a random known rating to fit our A and B matrices against. Then, starting on Line 7, for every 1000 iterations we calculate the RMSE to ascertain that the error is decreasing.

```
 1  ...
 2  While n < threshold && RMSE < thresholdRMSE
 3    x = randomKnownRatingCoordinates
 4
 5    SVD-algorithm(x)
 6
 7    if modulo(threshold,1000) == 0
 8      predictionMatrix = A * B
 9      predictionArray = extractValidationPredictions(predictionRating)
10      RMSE = sqrt(MSE(predictonArray,validationArray)
11
12      if RMSE < lowRMSE
13        lowRMSE = RMSE
14        thresholdRMSE = RMSE + 0.1
15      end
16    end
17    n += 1
18  end
```

Listing 4.1: Training of the prediction matrix

During training we needed to tune the learning rate η, regularization value λ, and the number of features f for each algorithm. A selection of these tests can be seen in Table 4.1 and Table 4.2.

We decided that a RMSE below 1 would be sufficient as the individual recommendation is not the main focus and we would not use too much time on parameter tuning. Based on the different setups we ended up using the Funk-SVD with the parameters in setup SA_4 from Table 4.1.

Setup	η	λ	f	RMSE
SA_1	0.001	0.01	100	1.838
SA_2	0.01	0.01	100	1.283
SA_3	0.01	0.2	100	1.029
SA_4	0.01	0.2	50	0.994

Table 4.1: Parameter tuning of Funk-SVD

Setup	η	λ	f	RMSE
SB_1	0.01	0.01	100	1.387
SB_2	0.05	0.01	100	1.006
SB_3	0.01	0.2	10	1

Table 4.2: Parameter tuning of SVD

4.3 Ranking

Prior to making a group recommendation, we needed to know each member of the groups favorite items in a ranked order. This is why we made the individual recommendations earlier. A users ranked list consists of their highest rated item by descending order. We get the items from the recommendation matrix where we select a users top-10. It is an arbitrary number, but it is a common standard in the recommendation field, so we choose it as well, but for future convenience it can be changed.

4.4 Rank Aggregation

Having a ranked list for each user in a group, it is then possible to aggregate them together in order to get a group recommendation. With basis in the voting approach BC and together with the basic BC method, which can seen represented in pseudocode in Listing 4.2, we constructed different variations thereof.

4.4.1 Borda Count

In Listing 4.2 there is a pseudo implementation of our BC method. It takes a matrix for the top-k preferences of users in a group as input and returns an array of ranked items, which is the group recommendations.

We wanted to use the indexes in the top-k matrix as our scoring system. In order to do so we had to reverse the top-k matrix, as it is sorted in descending order, which results in a users top preference being at index 1, which subsequently only gives 1 vote instead of k votes at index k. Next we had to find each unique item across all items for the users since we need to calculate the score of each distinct item.

From Line 8-16 we then iterate through the list of unique items in order to calculate a score for each of them. We do so by going though each user, starting at Line 10, to see if the item is on their list and if so adds its index position to its score value. The scores are stored in an array where their index matches the item they represent in the list of unique items, this happens in Line 15.

When the score is found for each item we had to find the top-k we wanted to return. This is done from Line 18-22. MatLab have a function to find the largest item in a array which can return both the value and index. We iterated though the list of scores to find the index of the largest score. Each time we iterated though, we would use the index to lookup items and add them to the result list and we then would set the largest score to be zero in order to find the next score.

```
 1 input: A matrix topK containing users individual ranked preferences.
 2 output: An array with the recommendations for the group in a ranked order.
 3 
 4 topK = reverse(topK)
 5 items = unique(topK)
 6 users = number of rows in topK
 7 
 8 for i = 1 to size(items)
 9   score = 0
10   for j = 1 to users
11     if movie(i) \in topK(j,:)
12       score += position of item(i) in array topK(j,:)
13     end
14   end
15   scores(i) = score
16 end
17 
18 for i = 1 to returnSize
19   I = index of highest value in scores
20   scores(I) = 0
21   result(i) = movies(I)
22 end
23 
24 return results
```

Listing 4.2: Borda Count implementation

4.4.2 Borda Weighted Count

BWC is almost identical with the BC in Listing 4.2. The only difference is that in Line 12 we, in BWC, adds an additional weight to the score. This weight is the number of times the item, i, occurs in the top-k matrix in order to give more frequent items a higher score.

4.4.3 Borda Transferable Count

The last method to be tested is BTC. This is a somewhat more comprehensive modification of the BC method with the base implementation being identical to BC, because the initial scores are assigned the same way as can be seen in Listing 4.3. The difference between this version and the original is that instead of ignoring the votes for items that did not get selected, this version redistributes them to see if it would make a difference on the return list, as inspired by STV, as explained in Section 2.4.3.

The first differences is that we rename the top-k list to topKItems and then create a new matrix over the vote distribution from the users in Line 4-9. Then everything is as usual until we reach Line 21. Here we created a loop that for each item i in items list we call the transferVotes function, which can be seen in Listing 4.4.

```
 1 items = unique(topK)
 2 users = number of rows in topK
 3 topKItems = revert(topK)
 4 topKVotes = size(topKItems);
 5 for i=1:row
 6     for j=1:col
 7         topKVotes(i,j)=j;
 8     end
 9 end
10
11 for i = 1 to size(items)
12   score = 0
13   for j = 1 to users
14     if movie(i) is in topK(j,:)
15       score += position of item(i) in array topK(j,:)
16     end
17   end
18   scores(i) = score
19 end
20
21 for i = 1 to size(items)
22   [result, scores, items, topKItems, topKVotes] = transferVotes(result, scores,
       ↪ items, topKItems, topKVotes)
23 end
24
25 ...
```

Listing 4.3: Borda Transferable Count implementation

The transferVote function is where we redistribute votes. There are two scenarios in which we need to transfer votes. The first scenario is shown on Line 5 and states that if highestScore value exceeds a threshold the item related to that score is added to the result list. We then find the difference between the highestScore and the threshold and save it in the transfer variable because these votes needs to be redistributed in order for them not to be wasted. In Lines 10-15 we find the list of users that voted on the highScore item called votingUsers, together with what they voted on it in list usersVotes. Next in Lines 17-23 we make the redistribution for the votes by iterating though votingUsers in order to find the items each user want to vote on instead. We select this item strategically by choosing the item closest to the threshold instead of voting on a users favorite item. We do this based on a theory that getting as many of their items as possible recommended to the group, gives more satisfaction than just getting their favorite item through. When we have found the item on a users top-k which is closest to reaching the threshold, we update the items score in Line 21 by adding a percentage of the redistribution votes based on how much the user voted on the highstScore item compared to the summation of all the users votes. After updating the score, the topKVoting matrix is updated with the redistributions. Lastly, the highestScore item is removed from the scores and items list.

The second case is if no item reaches the threshold. In this case we select the lowest scored item and redistributes its votes in the same manner as above and remove the item. We continue switching between these cases until we have the items we need either by having reached the return size with items above the threshold or until the combined size of the result and items list is the same as the return size.

```
users = rows in topkItems
threshold = summation of all votes divided with the return size
highestScore = highest value in scores

if highestScore > threshold
  result = highestScore
  transfer = highestScore - threshold
  votingUsers = []
  usersVotes = []
  for i = 1 to users
    if they voted on highestScore
      votingUsers += users(i)
      usersVotes += highestScore index in topKVotes
    end
  end

  for i = 1 to size(votingUsers)
    for j = 1 to column of topKItems
      highest = the user item with most votes in the group
    end
    scores(index of highest) += transfer * (usersVotes(i)/sum(votingUsers))
    update topKVoting with the new vote distribution
  end

  scores(index of highestScore) = [];
  items(index of highestScore) = [];

elseif size(result) + size(items) > returnSize
  This part is almost as of highestScore just for the lowestScore.
end
```

Listing 4.4: Implementation for the transfer method

4.4.4 Borda Escalating Count

As BWT, BEC, is very similar to BC, as can be seen in Listing 4.5. In BEC we split the items into three categories which is bot, mid and top. We define bot and mid in Line 9 and 10. Bot gets the size equal to $\frac{1}{3}$ of column size of top-k, mid gets the items from $\frac{1}{3}$ to $\frac{2}{3}$, and top is not defined, as it is those above the mid. We use these values to assign an additional weight to each user's items depending on their rank. In Line 19 one can see that a weight of 3 is added to items with index above the middle threshold. If the index is between the mid and bot thresholds it gets the weight of 1 and nothing if it is below bot threshold.

```
...
k = number of columns in topK
bot = k/3
mid = bot + k/3

for i = 1 to size(items)
  score = 0
  for j = 1 to users
    if movie(i) is in topK(j,:)
      index = position of item(i) in array topK(j,:)
      score += index
      if index > mid
              score = score + 3
            elseif index > bot and mid < index
              score = score + 1
           end
    end
  end
  scores(i) = score
end
...
```

Listing 4.5: Borda Escalating Count implementation

4.5 Concerns

During the implementation process some concerns surfaced, which we chose to ignore at that given moment for the sake of simplicity. The concerns regard how we scale with bigger groups and top-k items, and how a real world implementation should handle new users and items.

4.5.1 Scalability

Scalability was not a major focus point for us while we implemented our algorithms but it has not yet proven to be a problem with the setup we use.

Scaling could probably become a problem, if we were to look at a higher top-k than 10. We have a rather high worst-case runtime on our algorithms. BC, BWC, and BEC have a worst-case of $O(n \times m)$ and BTC has $O(n \times m \times v \times k)$ where n are the individual items, m is the group size, v is a subset of m who voted on a certain item, and k is the top-k size. Lets say that we have a top-k of 10 and a group size of 40 and there are no overlaps then n would be 400 items long but if top-k were 50 then n would be 2000. So for BTC we would have $2000 \times 40 \times 1 \times 50 = 4.000.000$ iterations of various loops.

4.5.2 Real World Application

For the sake of the development, testing, and evaluation the model-based recommendation in the form of SVD is very appropriate as we only have to train it once and then we have all the predictions.

If the group recommendation setup were to be used in real world applications, as the virtual reality example with random users described in Section 1.4, using SVD alone would be a cumbersome task due to the frequently new released items and new users, which would require us to retrain our model. A real world version should be more flexible so it would be able to handle changes to the user and item base.

One approach that could solve this is switching to a memory-based recommender method like Pearson correlation or a nearest neighbor approach. This would make a more flexible recommender system that would be able to handle the new items and users. Although this solves the flexibility problem it presents us with other problems. One of these problems is how to handle scalability with large amounts of data[31]. Predictions with these approaches are done online, so whenever a user needs recommendations they have to be calculated first. The calculation requires similar users to one receiving them, which requires making similarity calculations on users in a database, which can be a comprehensive task on large amounts of data.

Another approach would be a combination of a model- and memory-based methods. This could as an example be using the SVD, as we do now, and then if a new user should appear use a nearest neighbor approach to find a similar user in the model and use their recommendations instead. This approach still leaves us with the problem of new items.

5 Evaluation

5.1 Evaluation Setup

The five selection methods BC, BTC, BWC, BEC, and Random Selection(RS) are evaluated using the following setup. In total 6000 random groups were constructed:

- 1000 with 4 members
- 1000 with 8 members
- 1000 with 12 members
- 1000 with 16 members
- 1000 with 20 members
- 1000 with 40 members

The group sizes were decided upon based on our scenario, starting with a size of 4, which was the lower bound for what we called a group. Having a gradual increase in the size provide for convenient comparisons, while we wanted an extra large group size to see what would happen as the groups got larger. 40 was the largest logical group size to make, as it is both ten times the smallest, and each group still spans less than 5% of the total users in the dataset. Having 1000 groups of each size makes for a good statistical number, as the influence of individual outlier groups, whether they have exceptionally high or low satisfaction, are reduced to a state where a single group would be insignificant.

Additionally, it was decided to use a k of 10, so each members top-10 list is used. This number was decided arbitrarily, but as will be presented in Section 5.2.1 it happens to be a local maximum for the two methods we want to compare the most.

With the group sizes and k decided upon, what remains is the method of evaluation. nDCG will be the measure used, as discussed in 2.3.2, and it will be used on the individual groups. An average of the nDCG value is then taken for each group size. All the methods will be tested this way using the same groups and the same k.

5.1.1 Limitations of the Setup

First and foremost the setup is designed for the specific dataset used, and while methodology is applicable to other datasets, given a proper format of said datasets, the results gained through use of this specific setup on the dataset in question will only hints to the performance of the tested algorithms in general. Therefore, the evaluation of the algorithms should be seen as preliminary. With that said we will still give a preliminary conclusion based on the results gained through the evaluation.

Testing only certain group sizes might not give the full picture of the performance, however this is a preliminary evaluation, and as such the interesting results would be general trends rather than specifics. Similarly, the groups are constructed randomly, with no regard as to how similar or dissimilar the users are from one another. Both of these limitations could be improved upon or changed in a in-depth evaluation of the algorithms.

5.2 Satisfaction Evaluation

To test how the implemented methods performed compared to each other, the setup from Section 5.1 was used. Be aware that the scales of the figures differ a bit due to the random selection performing much worse than the other approaches thus dragging the lower end of the scale down.

Figure 5.1 presents the average satisfaction when the groups consists of 4 random members. Here there is only little difference in the fond satisfaction with BEC edging out slightly ahead, being a little less than 1% point better than BC. BTC and BWC are both minimally worse than BC, but as both the methods are more reliant on items being present in multiple top-k lists amongst the users, it is no surprise that they give worse results than BC.

Already when the group size is increased to 8, as seen in Figure 5.2, a trend of BTC outperforming the other methods start emerging, and at 8 members it outperforms BC by about 1.5% points. An interesting thing to note is that there is a generally sharp drop off in satisfaction for all methods when going from 4 to 8 members, but as the group size is increased, all but BTC drop further, while BTC maintains a satisfaction of around 0.59 until the group size is increased to 40.

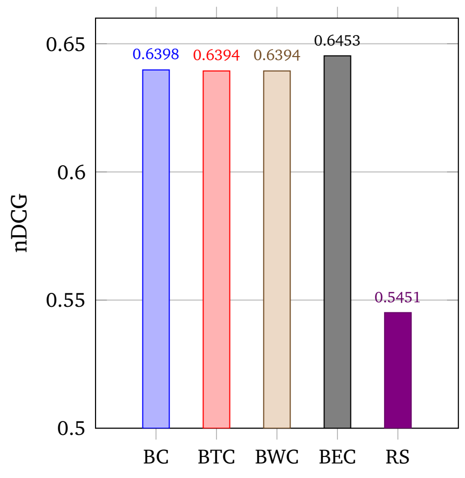


Figure 5.1: Groups with 4 users

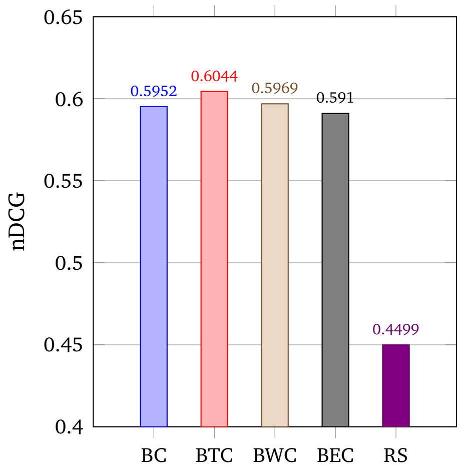


Figure 5.2: Groups with 8 users

BTC ability to satisfy larger groups becomes truly apparent with a group size of 12 and 16, Figure 5.3 and Figure 5.4, where it is about 6.4% points and 11.2% points better respectively. An interesting observation is that BTC gives a slightly worse satisfaction with a size of 12 than when the size is 16. The cause of this dip in satisfaction is unknown, however it could be interesting to explore it further. BC, BWC, and BEC all perform within 1% point of each other, with BEC performing slightly worse than the other two.

When the group size reaches 20, Figure 5.5, BTC also outperforms the other methods the most, at least with the group sizes tested. On average it outperforms BC by about 13.1% points. However after size 20, the benefits of BTC seems to stagnate, as Figure 5.6 shows, at a group size of 40 BTC had a slightly larger fall than the other methods, performing 12.9% points better than BC.

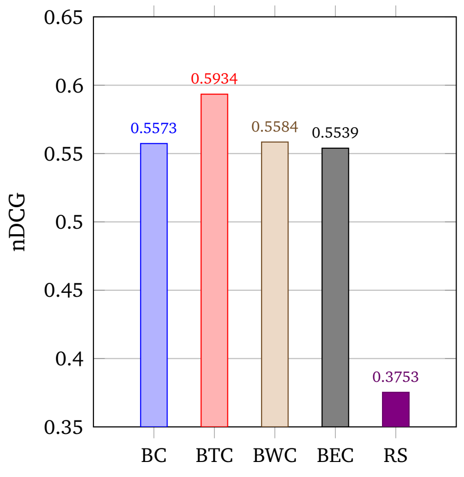


Figure 5.3: Groups with 12 users

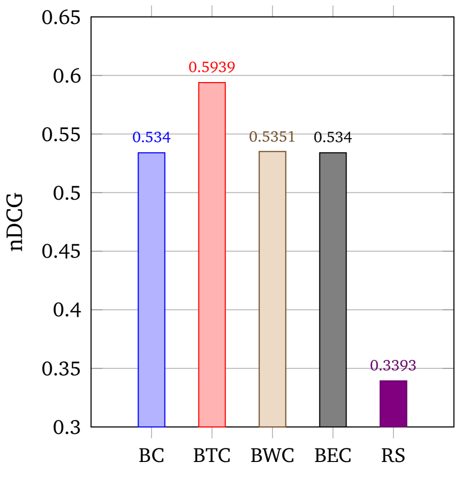


Figure 5.4: Groups with 16 users

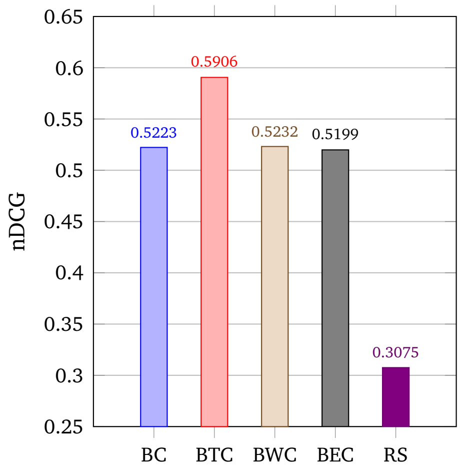


Figure 5.5: Groups with 20 users

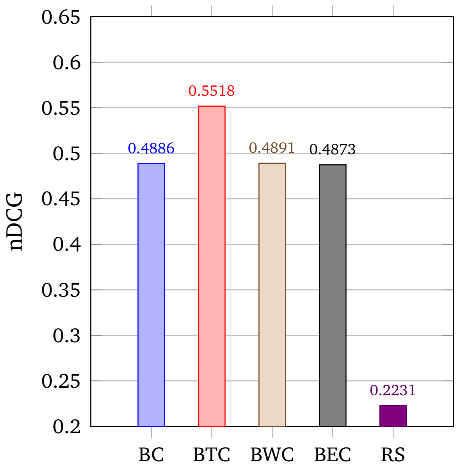


Figure 5.6: Groups with 40 users

5.2.1 Influence of the Size of k

While performing the satisfaction tests, an interesting observation regarding the size of k was made. We tested BC and BTC with a few alternative k, and found that as k increased from 10 to 20, there would be a drop in satisfaction. However at a k of 50, this drop off had been more than made up for, with a satisfaction about 10% points higher than the satisfaction at $k = 10$. Increasing k further to 100 also increases the satisfaction. These observations were found on tests performed on group sizes 16 and 40, other sizes were not tested.

During these prodding tests, it was discovered that there seemed to be a local maximum around a k of 10 or 11. These specific trends might be dependent on the dataset, as the only plausible explanation we could find, would be that the top-10 lists of each group holds comparatively more overlaps than the top-20 lists. For instance there might be 10 overlapping items in the top-10, while the top-20 only adds an extra 5 items with overlap, thus decreasing the percentage of items with an overlap in the set of items being evaluated upon. Similarly, top-50 might add a large number of new overlaps, increasing the final satisfaction, generally popular items could be located amongst items 21-50 in most users top-k lists.

5.2.2 Evaluation of Results

Based on the results presented in Section 5.2 it seems that BTC performs remarkably well on group sizes of more than 8. If the results for group sizes 20 and 40 are any indication, then BTC is about 13% better at satisfying a large group, when giving them 10 ordered items. On a group size of 4, BTC performs similarly to the other methods however slightly worse. Therefore, BTC should be used over BC when dealing with groups of size 8 or more.

As indicated by the findings presented in Section 5.2.1, which value of k that is used has little influence on which aggregation that performs best. However it has a large effect on how much satisfaction that is achieved overall. Ideally, for the MovieLens dataset, k should be 10 if the recommender should return a small number of items, otherwise it would be best to have it at more than 50. Whether this is generally applicable remains to be seen.

These results are a reflection of the evaluation performed on the MovieLens dataset, therefore, as mentioned in Section 5.1.1, the results should be tested on other datasets before anything can be concluded about the general application of the algorithms. That being said BTC shows promise and further study of the approach would be in order.

6 Conclusion

We wanted to showcase an approach that reconciles the differences in preferences among multiple group members in the problem of group recommendation. During this process we found that the evaluation possibilities of the performance of these approaches were rather limited due to lack of proper datasets making it difficult to determine whether or not a group would be satisfied with its recommendations.

We found that with using a ranked list for the recommendations we could adopt the normalized Discounted Cumulative Gain(nDCG) technique from the information retrieval domain to measure user satisfaction by comparing the ranked list of a user with the ranked list found through aggregation. This only left us with the task of forming the groups, which we ended up doing using random sampling.

We sought to answer the following questions from the problem statement:

- How to reflect group decision making algorithmically while securing satisfaction for the individuals as well as the group as a whole?
- How to measure the level of satisfaction in a group in regards to the items recommended?
- How to make recommendations to a group of people based on the users' individual preferences?

For reflecting how a group decision would be handled, we ventured into some of the methods used for resolving differing opinions. In the end, we incorporated single transferable vote into the Borda Count method in the Borda Transferable Count(BTC) algorithm. These are not methods that necessarily reflects an organic group decision process, but reflects parts of real group decision methodologies.

For measuring the level of satisfaction in a group, we found and used nDCG. It is commonly used for rankings and information retrieval. This works well with the view of recommendation as a tool for drawing out relevant information, when getting an overview is hard.

For making recommendations to a group, we present the BTC as a method of aggregating the individual ranked lists. The results are promising, as it performs better than the tested methods on nDCG scores. However, we cannot as of yet make a definite conclusion on the BTC as a solution to the group recommender problem.

6.1 Discussion

As mentioned we use the information retrieval method nDCG to measure satisfaction. In doing so we make several assumptions about the users' preferences. First of all, the user ratings used for the group recommendations are predictions, which may not coincide with the actual preferences of the user. Based on these predicted ratings we assume that a user would construct a ranked list, with a fixed distance between the top ranked and next highest rank regardless of the predicted preference for either, which we use to determine the satisfaction score, with the highest ratings. This means that the satisfaction score we get is based on assumptions about a user's preferences, which makes our results noteworthy, but ultimately, based on assumptions about existing data rather than real world data.

In order for us to get a concrete result we need to measure satisfaction with real users as to prove the effectiveness of the method presented in this project. Provided that, the method could point in the direction that group recommendation is a voting problem.

7 Future Work

7.1 Thesis Questions

Throughout the work done during this project, we have encountered or discovered a number of hurdles, possible improvements, and new approaches. These findings should be studied further and we will in our next project look into some of these.

Reordering of Rank Lists

While discussing ranked aggregation methods, one of the approaches we discussed but ended up not exploring is a method that involved re-ranking every users top-k based on the average value for the individual items amongst the other group members. For instance user 1's highest rated item only has the fifth highest average amongst the rest of the group members, so the item will be placed in user 1's ranked list as their fifth item. The idea behind this is to account for other users opinion (ratings) even if the item in question was not on the secondary users' top-k list. It is our hypothesis that making a re-ranking of the top-k list using other members' influence would lead to a better group satisfaction.

(1) To what extend does re-ranking a user's top-k preferences, based on other group members, influence the individual user as well as the groups overall satisfaction?

Obtaining Dataset

When we trained and tested our recommender, we only did so on the MovieLens dataset. While the results should be generally applicable, that might not be the case in practice, therefore the recommender and the aggregation methods should be tested on other datasets. In particular tests on a dataset with already existing groups and their satisfaction would be an welcome addition. However to the best of our knowledge, such a dataset with public availability does not exist.

(2) How does Borda Transferable Count perform in tests on a dataset designed for testing group recommendation?

This is further discussed in Section 7.2.

Alternative Rank Aggregations

As mentioned in Section 4.4.3 the implemented version of the Borda Transferable Count is not exactly as described in Section 3.1.3, due to the strategic transfer of the votes instead of as usual to the highest priority. While the implemented BTC seems to perform well compared to the methods we tested it against, ideally it should be further compared to other strategies, most prominently Spearmans Footrule and a BTC without the strategic reassignment.

(3) How does Borda Transferable Count with strategic reassignment perform compared to the state-of-the-art ranked aggregations?

7.2 Obtaining a Dataset

There is currently a lack of publicly available datasets for group recommendation.

We have requirements as to what makes an ideal dataset for our problem domain. The bigger the dataset the better for accurate learning and for splitting the data into subsets for validation. Datasets such as the HappyMovie dataset is too small, but presents many of the necessary traits of a good group recommender dataset[30].

It needs users and groups, and there are several options to go with here. The groups could be ephemeral, persistent, or anything in between, but it has an effect on gathering data, as one unconsciously affect the data. A truly random selection of people for a group could introduce obstacles uncommon for a group of friends, such as sharing no language common to all for a movie recommendation, which could become the main challenge of a good recommendation to the exclusion of individual taste for such a configuration. Such a problem would not be addressable depending on how the groups are formed. Having many groups of various sizes would be preferable, as group recommender systems perform differently on various group sizes. It is not a problem whether users shows up in more than one group. It is a good cost-saving measure to get more data out of fewer people. The data should have ratings of items in such a way that individual preferences can be inferred. However, the most important part are the group recommendations. It would be ideal to have groups of users make an individual ranked list each and one for the group together. This could be used to both measure the satisfaction of the users and also give us an idea of how well we reflect the decision making process.

Assuming a new dataset will not become available, there are two ways for us to make our own. However, the necessary scale in order to get a dataset of the right size makes it a difficult task to do in an inexpensive manner.

The hardest option is creating or convincing a popular service provider in the domain of group recommendation to help provide data of user behavior. Netflix is an example of a service provider who leveraged their position to create a dataset. A survey could also be launched and spread through social media, with the trade between survey and user being behavioral data in return for useful insights about habits or traits, but it relies on the survey going viral, which is an unreliable path.

The alternative is gathering the data for the sole purpose of making the dataset, which is the most straightforward. We have two options for approaching this, finding willing participants for a survey or crowdsourcing.

Finding enough willing participants is the main problem, as the dataset needs, at the lowest, enough to form several hundred groups. Though there are more options than ever through services like Mechanical Turk by Amazon, crowdsourcing presents its own challenges. Participants here are paid for their time and effort in the order of 10 to 50 cents per user, which is a reliable way of motivating participation, but makes a large dataset expensive. Another challenge is introducing the group decision aspect, as known crowdsourcing sites do not have the framework for groups of workers to cooperate on a task simultaneously, rather than independently, and reach a consensus before submitting results. There are possible ways to offset this, such as simulating group recommendations and survey the user on their satisfaction with the results, but it definitely requires careful consideration of how to proceed in order to get useful data.

7.3 Influence and Context

Quintarelli et al documented the positive effect of influence and contextual influence when recommending items for a group[21]. An example the article uses to explain contextual influence with, is the scenario of a family, consisting of parents and kids, selecting what to watch on the television. In this scenario they have two contexts, early afternoon and late afternoon. They then argue that in the early afternoon the children have a greater influence because most of the tv-programs are children friendly at this time. In the late afternoon the parents then get more influence because child inappropriate programs may occur.

The idea of context influence could be applicable in many scenarios and the selection can be extensive. It could be interesting to apply this approach to our method in order to see if it would improve the user satisfaction, but in order to do this we would need a dataset over group preferences in order to detect any improvements. Furthermore, the dataset should contain some sort of context.

Bibliography

[1] Kenneth J. Arrow. "A Difficulty in the Concept of Social Welfare". In: *Journal of Political Economy* 58.4 (1950), pp. 328–346. ISSN: 00223808, 1537534X. URL: http://www.jstor.org/stable/1828886.

[2] Linas Baltrunas, Tadas Makcinskas, and Francesco Ricci. "Group recommendations with rank aggregation and collaborative filtering". In: *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010*. Ed. by Xavier Amatriain et al. ACM, 2010, pp. 119–126. ISBN: 978-1-60558-906-0. URL: http://doi.acm.org/10.1145/1864708.1864733.

[3] Luis M. de Campos et al. "Managing uncertainty in group recommending processes". In: *User Modeling and User-Adapted Interaction* 19.3 (2009), pp. 207–242. ISSN: 1573-1391. URL: http://dx.doi.org/10.1007/s11257-008-9061-1.

[4] Lucas Augusto Montalvão Costa Carvalho and Hendrik Teixeira Macedo. "Users' satisfaction in recommendation systems for groups: an approach based on noncooperative games". In: *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*. Ed. by Leslie Carr et al. International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 951–958. ISBN: 978-1-4503-2038-2. URL: http://dl.acm.org/citation.cfm?id=2488090.

[5] O. Celma. *Last.fm Dataset*. 2010. URL: http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html.

[6] Ingrid A. Christensen and Silvia Schiaffino. "Entertainment recommender systems for group of users". In: *Expert Systems with Applications* 38.11 (2011), pp. 14127 –14135. ISSN: 0957-4174. URL: http://www.sciencedirect.com/science/article/pii/S0957417411007482.

[7] Amra Delic et al. "Observing Group Decision Making Processes". In: *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*. Ed. by Shilad Sen et al. ACM, 2016, pp. 147–150. ISBN: 978-1-4503-4035-9. URL: http://doi.acm.org/10.1145/2959100.2959168.

[8] Cynthia Dwork et al. "Rank aggregation methods for the Web". In: *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*. Ed. by Vincent Y. Shen et al. ACM, 2001, pp. 613–622. ISBN: 1-58113-348-0. URL: http://doi.acm.org/10.1145/371920.372165.

[9] Daniel Fleder and Kartik Hosanagar. "Blockbuster Culture's Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity". In: *Management Science* 55.5 (2009), pp. 697–712. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.151.3210.

[10] Gene Golub and William Kahan. “Calculating the singular values and pseudo-inverse of a matrix”. In: *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2.2 (1965), pp. 205–224.

[11] F. Maxwell Harper and Joseph A. Konstan. “The MovieLens Datasets: History and Context”. In: *TiiS* 5.4 (2016), p. 19. URL: http://doi.acm.org/10.1145/2827872.

[12] Yehuda Koren, Robert Bell, Chris Volinsky, et al. “Matrix factorization techniques for recommender systems”. In: *Computer* 42.8 (2009), pp. 30–37.

[13] Daniel D. Lee and H. Sebastian Seung. “Algorithms for Non-negative Matrix Factorization”. In: *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*. Ed. by Todd K. Leen, Thomas G. Dietterich, and Volker Tresp. MIT Press, 2000, pp. 556–562. URL: http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.

[14] Daniel D Lee and H Sebastian Seung. “Learning the parts of objects by non-negative matrix factorization”. In: *Nature* 401.6755 (1999), pp. 788–791.

[15] Judith Masthoff. “Group Modeling: Selecting a Sequence of Television Items to Suit a Group of Viewers”. In: *User Modeling and User-Adapted Interaction* 14.1 (2004), pp. 37–85. ISSN: 1573-1391. URL: http://dx.doi.org/10.1023/B:USER.0000010138.79319.fd.

[16] The Electoral Knowledge Network. *Single Transferable Vote*. 2016. URL: http://aceproject.org/main/english/es/esf04.htm.

[17] David L. Poole and Alan K. Mackworth. “Artificial Intelligence. Foundations of computational agents”. In: New York, NY 10013-2473, USA: Cambridge University Press, 2010, pp. 149–151. ISBN: 9780521519007.

[18] David L. Poole and Alan K. Mackworth. “Artificial Intelligence. Foundations of computational agents”. In: New York, NY 10013-2473, USA: Cambridge University Press, 2010, pp. 304–306. ISBN: 9780521519007.

[19] David L. Poole and Alan K. Mackworth. *Artificial Intelligence. Foundations of computational agents*. New York, NY 10013-2473, USA: Cambridge University Press, 2010. ISBN: 9780521519007.

[20] George Popescu. “Group Recommender Systems as a Voting Problem”. In: *Online Communities and Social Computing: 5th International conference, OCSC 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013. Proceedings*. Ed. by A. Ant Ozok and Panayiotis Zaphiris. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 412–421. ISBN: 978-3-642-39371-6. URL: http://dx.doi.org/10.1007/978-3-642-39371-6_46.

[21] Elisa Quintarelli, Emanuele Rabosio, and Letizia Tanca. "Recommending New Items to Ephemeral Groups Using Contextual User Influence". In: *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*. Ed. by Shilad Sen et al. ACM, 2016, pp. 285–292. ISBN: 978-1-4503-4035-9. URL: http://doi.acm.org/10.1145/2959100.2959137.

[22] Francesco Ricci et al. *Recommender Systems Handbook*. Second Edition. New York, NY, USA: Springer-Verlag New York, Inc., 2015. ISBN: 9781489976369.

[23] Francesco Ricci et al. "Recommender Systems Handbook". In: Second Edition. New York, NY, USA: Springer-Verlag New York, Inc., 2015. Chap. 22. ISBN: 9781489976369.

[24] Francesco Ricci et al. "Recommender Systems Handbook". In: Second Edition. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 228–236. ISBN: 9781489976369.

[25] Francesco Ricci et al. "Recommender Systems Handbook". In: Second Edition. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 85–86. ISBN: 9781489976369.

[26] Francesco Ricci et al. "Recommender Systems Handbook". In: Second Edition. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 321–322. ISBN: 9781489976369.

[27] Francesco Ricci et al. "Recommender Systems Handbook". In: Second Edition. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 282–283. ISBN: 9781489976369.

[28] Francesco Ricci et al. "Recommender Systems Handbook". In: Second Edition. New York, NY, USA: Springer-Verlag New York, Inc., 2015, p. 290. ISBN: 9781489976369.

[29] Francesco Ricci et al. "Recommender Systems Handbook". In: Second Edition. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 750,753. ISBN: 9781489976369.

[30] Lara Quijano Sánchez, Juan A. Recio-García, and Belén Díaz-Agudo. "HappyMovie: A Facebook Application for Recommending Movies to Groups". In: *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*. IEEE Computer Society, 2011, pp. 239–244. ISBN: 978-1-4577-2068-0. URL: http://dx.doi.org/10.1109/ICTAI.2011.44.

[31] J. Ben Schafer et al. "Collaborative Filtering Recommender Systems". In: *The Adaptive Web, Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Vol. 4321. Lecture Notes in Computer Science. Springer, 2007, pp. 291–324. ISBN: 978-3-540-72078-2. URL: http://dx.doi.org/10.1007/978-3-540-72079-9_9.

[32] Shilad Sen et al., eds. *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*. ACM, 2016. ISBN: 978-1-4503-4035-9. URL: http://doi.acm.org/10.1145/2959100.

[33] Electoral Reform Society. *What is STV?* 2010. URL: http://www.electoral-reform.org.uk/sites/default/files/What-is-STV.pdf.

[34] Lawrence E. Spense, Arnold J. Insel, and Stephen H. Friedberg. “Elementary Linear Algebra. A matrix approach”. In: 2nd. Upper Saddle River, NJ 07568: Pearson Education, Inc., 2008, pp. 438–446. ISBN: 0131580345, 9870131580343.