UPPSALA
UNIVERSITET

UPTEC IT 20021
Examensarbete 30 hp
Juni 2020

Switching hybrid recommender system to aid the knowledge seekers

Alexander Backlund

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Switching hybrid recommender system to aid the knowledge seekers

Alexander Backlund

In our daily life, time is of the essence. People do not have time to browse through hundreds of thousands of digital items every day to find the right item for them. This is where a recommendation system shines. Tigerhall is a company that distributes podcasts, ebooks and events to subscribers. They are expanding their digital content warehouse which leads to more data for the users to filter. To make it easier for users to find the right podcast or the most exciting e-book or event, a recommendation system has been implemented. A recommender system can be implemented in many different ways. There are content-based filtering methods that can be used that focus on information about the items and try to find relevant items based on that. Another alternative is to use collaboration filtering methods that use information about what the consumer has previously consumed in correlation with what other users have consumed to find relevant items. In this project, a hybrid recommender system that uses a k-nearest neighbors algorithm alongside a matrix factorization algorithm has been implemented. The k-nearest neighbors algorithm performed well despite the sparse data while the matrix factorization algorithm performs worse. The matrix factorization algorithm performed well when the user has consumed plenty of items.

Handledare: Antonie Hedgecock, Emil Paulsson
Ämnesgranskare: Johan Alenlöv
Examinator: Roland Bol
ISSN: 1401-5749, UPTEC IT 20021

Sammanfattning

I vårt dagliga liv är tiden väsentlig. Människor har inte tid att filtrera och välja bland de hundratusentals digitala val som de ställs inför. För att underlätta detta används rekommendationssystem som underlättar filtreringen för användarna. Tigerhall är ett företag som distribuerar podcast, e-böcker och evenemang till prenumeranter. De håller på att utökar sitt digitala lagerhus med fler podcasts, e-böcker och evenemang vilket leder till mer data för användarna att filtrera. För att göra det lättare för användaren att hitta den rätta podden eller den mest spännande e-boken eller den lämpligaste evenemanget så har ett rekommendationssystem implementerats. Ett rekommendationssystem kan implementeras på flera olika sätt. Det finns innehållsbaserad filtrerings metoder som använder sig av information om föremålen för att hitta relevanta föremål. Ett annat alternativ är att använda sig av gemensam filtrerings metoder som använder sig av information om vad användaren har konsumerat tidigare i samspel med vad andra användare har konsumerat för att hitta relevanta föremål.

I detta projekt har ett rekommendationssystem utvecklats. Detta system använder sig av två olika algoritmer, den ena är en K-närmaste grannar-algoritm och den andra är en matrisfaktoriseringsalgoritm. K-närmaste grannar-algoritmen är en maskininlärnings algoritm där man försöker klassificera ett föremål till en större grupp. Både K-närmaste grannar-algoritmen och matrisfaktoriseringsalgoritmen är båda metoder av gemensam filtrering. Matrisfaktoriseringsalgoritmen går ut på att dela upp en större, glesare matris till två mindre matriser. Dessa matriser tränas först upp och sedan sammanförs till en gemensam matris igen, då med förutbestämda omdömen. Detta rekommendationssystem är därmed hybrid, då den använder sig av flera algoritmer.

K-närmaste grannar-algoritmen fungerade bra trots de glesa data som tillhandahölls medan matrisfaktoriserings-algoritmen presterade mindre bra. När användaren hade konsumerat många podcasts, evenemang och/eller e-böcker blev resultatet för matrisfaktoriseringsalgoritmen bättre.

Contents

1 Introduction **1**

1.1 Recommender system 1

1.2 Objective, research question and motivation 2

2 Related work **4**

2.1 Netflix 4

2.2 YouTube 6

3 Theory **7**

3.1 Interactive Solutions and Tigerhall 7

3.2 Collaborative filtering 9

3.2.1 Neighborhood-based algorithms 10

3.2.2 Matrix factorization 12

3.3 Content-based filtering 18

3.4 Hybrid recommender system 20

3.5 Evaluation types 20

3.5.1 Online metrics 21

3.5.2 Offline metrics 22

4 Implementation **26**

4.1 Application data 26

4.2 Recommender system 27

4.3 Mobile application 30

5 Results **32**

6 Discussion **39**

7 Conclusion **41**

8 Future work **42**

A Algorithms **45**

1 Introduction

We are victims of data overflow, there is too much data for a single user to process. That is why recommender systems have become very relevant in our daily life. The main purpose of a recommender system is to sift out the relevant information for a specific user. This basically means to personalize the users experience within a given system. Countless websites use recommender systems to make the visit as pleasant as possible for the users. If the user browses an online clothing store and buys a shirt, the website can prompt the user with a row with similar items that other users have bought which also bought the same shirt. This is one way of recommending items to the users and Amazon.com uses this way of recommending items [1]. Another way of recommending items to the user is to analyze the user's history to see what the user likes and dislikes. With this information the items can be ranked and presented to the user. One way Netflix recommends movies is by analyzing the user, and based on the user's history the user's dashboard may look different from other users' dashboards [2]. Netflix also uses rows to categorize the items and one of the rows has the title "Top picks for you" which is a collection of recommended items based on the user's history.

Another company that wants to implement a recommender system is Tigerhall. Tigerhall is a company that makes it easy for users to expand their knowledge by listening to inspirational podcasts, read eBooks, or attend events led by experts in specific fields. The experts are different CEOs, directors, founders, and so forth, an example is Gustavo Fuchs, General Manager at Microsoft another is Aniruddha Ganguli, Co-Founder CEO at Negobot. Tigerhall are expanding their products which lead to more data. The goal of this project is to help Tigerhall to personalize their selection of items for each user. This is tackled by developing a recommender system that recommends relevant podcasts, eBooks and events to the user based on previously consumed items and based on similar users.

1.1 Recommender system

We are surrounded by companies that store data about how and what we interact with on the Internet. In the book "Recommender systems - the textbook" it is stated that the user is a kind of catalyst for the development of recommender systems [3].

This can be explicitly done by having the user rate a certain object or implicitly done by logging what the user is looking at and endorsing those items [3, Chapter 1.1]. The amount of data that humans consume per day in form of newspaper, movies, TV shows, books, etc., have increased over the previous two decades and there is still more data to

consume the next day [1]. How do we know which type of data which is relevant? With the help of different kinds of recommender systems the relevant data will be shown to you. For example, if you one evening is watching an action movie on Netflix, the next evening there will be some new action movies recommended to you, this is done by the recommender system of Netflix [2].

A recommender system usually need some type of data to be able to function correctly. One way of collecting data from the users is by letting the user rate the item in the dataset which they have consumed. If we take Netflix as an example again, a user watch the movie ”Shrek” and rate it very high and then watch the movie ”Scream” and rates it very low, the recommender system now has explicit data from the user. Another way of collecting data can be to monitor the user and analyze what the user looks at or buys. This is an example of collecting data implicitly, the user doesn't have to do anything extra. This is done by Amazon.com [3, Chapter 1.1].

One major goal of a recommender system is to increase the profits of the company[3, Chapter 1.2]. If a user is presented with more relevant recommendations of what to buy after buying item A the profit will be increased. Take two different recommender systems A and B, if a user buys an iPhone X then recommender system A recommends that the user should buy a pair of Airpods too, but recommender system B suggests that the user should buy a book about horses. The company with the recommender system A would most probably have a higher profit. This is because giving relevant suggestions keeps the user interested and curious which may increase the profit drastically [2]. By always recommending relevant items to the consumers the customers' satisfaction will improve and so will their loyalty to the company [3, Chapter 1.2].

1.2 Objective, research question and motivation

Given the vast range of topics Tigerhall offers, how can a personalized representation of their content be achieved? If a user has only listened to podcasts about how to influence people and another user has only read eBooks about making money should their apps dashboard have the same appearance? Should they only be presented with the most popular podcasts which may not be in the same category as their content history? How can a relevant recommendation be done given the sparse data Tigerhall have? Given that the items per category is few how can the recommendations encourage the user to start consuming other items from other categories?

These are some of the questions this thesis has to face. The objective of the thesis is to investigate how the data of the application can be personalized for each user and give a more immersive experience. A machine learning and linear algebra approach

was taken to achieve this. Given the users' history and other users' taste a relevant recommendation is done to the user. Given the problem the following research question was made:

Given the sparse data and different types of items how can a relevant recommendation be accomplished which will encourage the user to continue using the application?

2 Related work

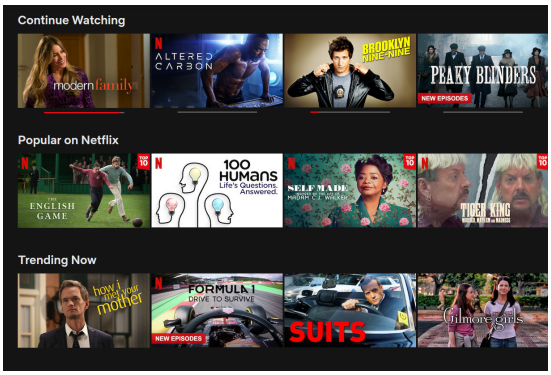
As mentioned in the introduction there exist many recommender systems. In this section both Netflix's and YouTube's recommender systems will be explained in more detail to get an deeper understanding of how recommender systems work. Netflix is a subscription streaming site where the user can watch movies without owning a copy of it, for more detail see Section 2.1. YouTube is also a streaming site but instead of showing movies YouTube lets the users upload videos and watch other users' videos, for more detail about YouTube see Section 2.2

2.1 Netflix

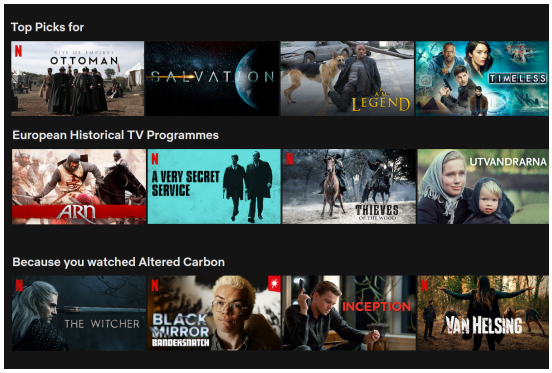
When talking about recommender systems one huge player is Netflix, they use several recommender algorithms to make their recommendations good. In the paper by Gomez and Uribe [2] the authors present all the algorithms Netflix uses to categorize all their movies and display the movies to the users in a personalized way. Here are the categories:

- Personalized Video Ranker (PVR): This algorithm orders a subset of the items filtered by genre or another filter, in this case movies, in a personalized way, see Figure 1b. The most relevant movie for the user will be displayed first in the row and then the second most relevant and so on. Because of the PVR, the same genre will have different items depending on the user.
- Top-N Video Ranker: While PVR is restricted to one type of filtering the Top-N algorithm produces personalized movies from the entire item pool. The Top-n approach can suggest items from different genres, see Figure 1b. Otherwise the two algorithms almost work the same using the personalizing with the popularity and looking at viewing trends.
- Trending Now: This algorithm looks at short-term trends, see Figure 1a. There are two different types of trends this algorithm looks at. The first one is when a trend repeats every month or year, for example users usually want to look at romantic movies close to Valentine's Day or Christmas movies close to Christmas. The second type of trend is when something happens in the world for example a new pandemic occurs and then the trend for disease movies or documentaries about pandemics increase. Both of these trends only have a short-term effect when they occur.

- Continue Watching: In this section only items that the user has watched previously is presented, see Figure 1a. The algorithm tries to determine if the recently watch movie is going to be watched again or if the user doesn't want to watch the movie/TV-show any more. The data given to the algorithm is the time since the user viewed the movie, at what point in the movie the user stopped watching (start, middle or end of the movie), what device was used, and if the user has watched anything else since.
- Video-Video Similarity: This is their most traditional recommendation category. It will present the title of the movie the user watched previously and then similar movies to that one, see Figure 1b. One example of this would be a category with the name "Because you watched The Lord of the Rings: The Two Towers" and the items in the category would be "The Lord of the Rings: The Return of the King", "The Hobbit: An Unexpected Journey" and so on.



(a) On top we can see the "Continue Watching" row where items which were not finished are. Under that the "Popular on Netflix" row is which displays popular items. Lastly the "Trending Now" row which will display short-term trends.



(b) The "Top picks" row is on top which will give a personalized subset from all the items for the user with the help of the Top-N video ranking algorithm. In the middle section we have the "European Historical TV Programmes" row which displays the most relevant items first in the given genre, this row uses the personalized video ranker algorithm. In the bottom row the "Because you watched..." which displays items similar to the given item in this case "Altered Carbon". This row uses the video-video similarity algorithm.

Figure 1: An overview of how Netflix present its items.

Netflix also uses recommendation algorithms to determine which kinds of categories will be displayed on the user's home screen. They also have algorithms that determine what kind of information should be displayed for each movie, lastly, they have a set of algorithms handling their search engine.

2.2 YouTube

The most popular video-sharing platform is called YouTube and has more than a billion users [4]. YouTube has a huge library of videos and are increasing the library with many hours of video every second which can lead to some problem when trying to find new and interesting videos [4, 5]. The authors of the report [5] continues by stating that YouTube is using a hybrid system for their recommendations which is implemented using deep learning neural network. The hybrid system looks at both explicit and implicit feedback from the users. YouTube gives the user the option to like or dislike a video and comment on the video which is a type of explicit feedback. YouTube also tracks how long a user has watched a certain video and if the user finishes it or not and so on, this is types of implicit feedback [5].

3 Theory

In this section a brief introduction to Interactive Solution and Tigerhall will be presented and also a deeper dive into the system architecture of the application. Following this, a more detailed explanation about how the recommender systems will be described. Recommender systems can usually be divided into three groups which are collaborative filtering (CF) systems (see Section 3.2), content-based filtering (CBF) systems (see Section 3.3) and the last one is a hybrid of the previous ones (see Section 3.4). This section will end with some theories about the evaluation of recommender systems.

3.1 Interactive Solutions and Tigerhall

Interactive Solutions is a software development consultancy firm stationed in Uppsala. Tigerhall is one of Interactive solutions customers and Interactive Solutions develop and maintain Tigerhalls mobile application.

Tigerhall is an Asian company founded in 2018. Their goal is to connect ambitious people with influential leaders either via personal events or offline through eBooks or podcasts. They try to develop users' leadership, increase the sales of companies, help users live a healthier life and more through knowledge of experts in the field.

Tigerhall uses several different services. Many services are for the developers and the administrators at Tigerhall such as error tracking and data visualization tools. All these services are displayed in Figure 2. The majority of the users will use the mobile application where the podcasts and eBooks can be consumed and events can be booked. In the Tigerhall backoffice, the administrators at Tigerhall can monitor the application. The data is also visualized and presented to the administrators in this part of the system. In the following list all external services are briefly explained:

1. Stripe - is an online payment system which Tigerhall uses to handle the payment of all their users. The users need to subscribe on Tigerhall to get all the content Tigerhall provides and Stripe handles the payments.

2. Insider - handles the push notification of the application. A push notification is a message which will be displayed on the user's mobile phone. Insider also handles the tracking.

3. Mailgun - handles all the emails which the administrators can send out to the users.

4. Sentry - is a bug reporting system. When a bug occurs a new ticket will appear in sentry, which provides information about the steps taken before the bug occurred and on what type of phone etc.
5. Elasticsearch - is a search engine which Tigerhall uses in the mobile application.
6. Kibana - is a front-end for the data acquired by Elasticsearch
7. Elasticsearch APM - Tigerhall uses Elasticsearch APM to monitor the performance of the search engine.
8. Grafana - is used by the administrators at Tigerhall to easily visualize data stored by the application. An example could be a graph of the most popular items consumed by the users within a given period.
9. Prometheus - is an alerting system that alerts both the developers at Interactive Solutions and the administrators at Tigerhall when the application is down or having a problem.
10. MinIO - is the file storage.
11. PostgreSQL - is the database used by Tigerhall.

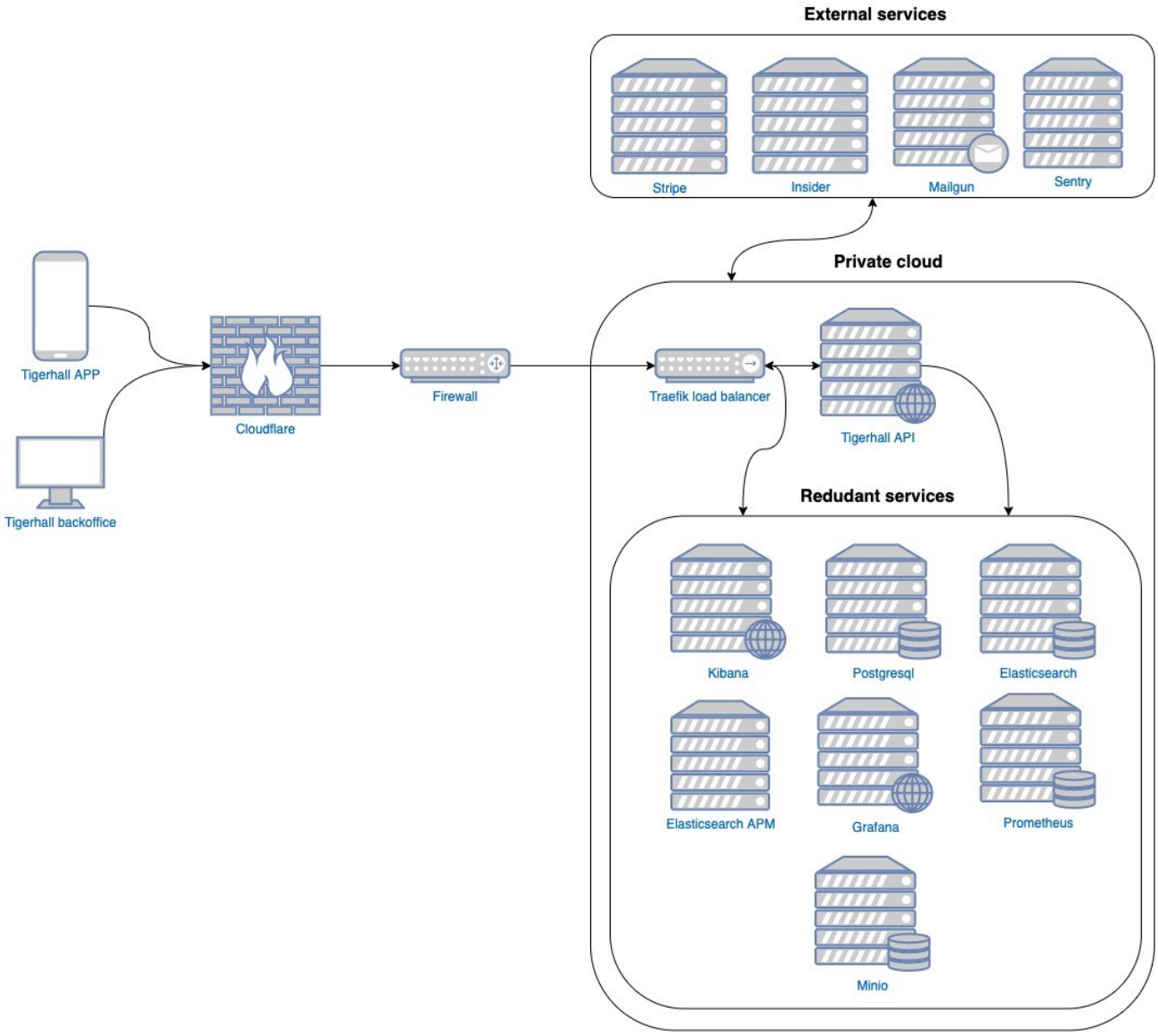


Figure 2: An overview of the system architecture

3.2 Collaborative filtering

Systems that use CF in their recommendation system use the power of multiple users ratings to make a recommendation. The availability of user ratings is of essence to implement a CF system. This type of recommendation system looks at a target user

and tries to find similarities between other users and the target user, and then it makes recommendations based on similar users' tastes. This is done without any knowledge about the item itself [6]. One example of this could be that we have two users Sebastian and Norah which both have rated movies the same way, the system will notice this and if Sebastian has rated one movie that Norah hasn't the system can guess how Norah would rate the movie.

One major benefit of using CF over CBF systems is that it can suggest relevant items that have no direct link to the user [7]. One drawback to CF is that when a new user starts using the system there are no prior ratings done by the user. This makes finding similar users with similar taste a challenge, the same goes for the items. If no user has rated the item then, in a CF system, that item will rarely be recommended. This is called the cold-start problem [6]. The authors of the report [8] stated that a common approach to tackle this problem is to use a CBF system or a hybrid recommender system with both CF and CBF recommendation. The hybrid version usually includes information about the items when recommending which makes a new item easy to recommend based on the metadata about the item. When a new item is introduced to the hybrid recommender system it can be recommended easily so the cold-start problem with items can be solved with a hybrid system. To solve the user cold-start problem, information about demography can be used to cluster the user with other users [8]. The best is to integrate both of these into the recommendation algorithm to make it more robust.

3.2.1 Neighborhood-based algorithms

There exist two different principles when talking about neighborhood-based methods. These two methods are item-based methods and user-based methods [3, Chapter 2.3]. Both of these methods are types of CF. The first method assumes that similar items are rated the same by the same user, then the rating of a new item which is similar to the previous items can be guessed[3, Chapter 2.3]. Given a user-item matrix, which is a matrix with all the users as rows and all the items as columns or vice versa. All the cells in the matrix are the rating of user i (where i is the row) on the item j (where j is the column). This approach analyzes the matrix to find relationships between the items. Using the relationships between the items a recommendation can be made. The motivation for the recommendations is that if a user consumes or buys an item then the user probably wants to consume a similar item [9]. For example if Emil has rated the godfather 1 and 2 high then one can predict that the rating of Goodfellas would also be high. This is an example of an item-based approach.

The second approach would be a user-based approach where the core is the similarities between users. An example of this would be if Sofia and Katarina have seen similar

movies in the past and rated them the same then one can use Sofia's rating on an item that Katarina has not yet seen. Here is a more detailed example of the user based approach for a recommendation.

	Movie 0	Movie 1	Movie 2	Movie 3
Katarina	2	3	5	?
Sofia	1	?	1	5
Emil	3	2	1	4

Figure 3: A matrix with users rating on different movies.

This example was inspired by the example in the book [3, Chapter 2.3.1]. In Figure 3 we can see three users' ratings on 4 different movies. Now we want to recommend a movie to the user Nils which have seen the movies 0 and 2 and rated them 2 and 4. This means that we need to calculate the predicted rating of the movie 1 and 3 and see which one would suit Nils best. This is done by calculating the similarities of Nils to all the other users. The similarity calculations are only done with the known data so Sofia's rating on movie 1 will not be included in the calculation. Many similarity measures can be used, in this example the cosine similarity (1) will be used. The cosine similarity is calculated on the items which the target user (Nils in this case) and all the other users have in common.

$$\text{Cosine}(x, y) = \frac{\Sigma_{i=1}^{n} x_i y_i}{\sqrt{\Sigma_{i=1}^{n} x_i^2}\sqrt{\Sigma_{i=1}^{n} y_i^2}}, \tag{1}$$

where x_i is the rating of user 1 and y_i the rating of user 2. By using Equation (1), the Figure 4 was created where the similarity between a user and Nils is presented. It is stated in the table that Katarina is the most similar to Nils. With the knowledge that Katarina and Nils have a similar taste, an item which Nils has not consumed but Katarina has and that Katarina rated high can be recommended. In this case, the movie 1 would be recommended.

	Cosine similarity
Katarina	0.996
Sofia	0.9486
Emil	0.70

Figure 4: The similarity score between Nils and all the other users.

In the user-based method the recommendations are often less accurate than in the item-

based method. This is because in the item-based method all the recommendations are based on the ratings given by the user itself and not other users [3, Chapter 2.3]. Also if one item is ranked high then a similar item is most probably rated high as well for the same user. In the user-based methods the similarities between the target user and the similar users depend on a common interest, but other users may have different interests which can lead to less accurate recommendations. This phenomenon can be an advantage for the user-based methods, because it gives more diverse recommendations and it also encourages serendipity. Serendipity can occur when a recommendation is positively received but came as a surprise to the user, or an item which was interesting for the user but not similar to anything other the user has consumed. This is something that the item-based methods will have trouble achieving because of the similarities in the recommendations. It is stated in the book by Aggarwal [3, Chapter 2.3] that "without sufficient novelty, diversity, and serendipity, users might become bored with very similar recommendations to what they have already watched". This can be a problem in only using the item-based method. In the same system, the recommendations are transparent and there is a clear motivation for the recommendations, for example, "Because you watch item A" the recommendations will contain three similar items to item A. This is done by Netflix, see Section 2.1 for more detail.

3.2.2 Matrix factorization

One goal of matrix factorization in the context of recommender systems is to be able to predict ratings on items which have not been rated before by a user. In other words, given a sparse matrix be able to determine unseen values in the matrix. In CF a user-item matrix is used where the rows are user and the columns are items. All the cells are the ratings of user i on item j, this matrix can be factorized into two or more matrices. One of the matrices is the user in the latent space and the other is the item in its latent space [10]. Figure 5 is a simple representation of a matrix factorization process, the left side represents the whole sparse user-item matrix and the right side represent the division of the user and the items into two different matrices where the rows are the users and the items and the columns are the latent factors for the users and the items. Latent factors are values that are inferred from other observed variables [11]. Then given a specific user i from the user matrix and a specific item j from the item matrix the rating can be estimated by taking the dot product of the row of the user i and the row of the item j. This will be an estimation of the rating. This will result in an optimization problem where the estimation of a rating given a user and an item should be as close to the actual rating. The goal is to minimize the error which is given from the real rating and the estimated rating. To minimize the error, gradient descent can be used. By using this method the latent factors will be updated so that the result from the dot product

between users and items will be closer to the real value. After the latent vectors have been updated over several iterations the user and item latent vectors will be optimized to represent the user or the item. The latent vectors can then be used on unseen items and give an estimation of how well a specific item would fit a specific user by doing the dot product of the two vectors.

The algorithm that is used to perform the matrix factorization follows the steps which is outlines in [3, Chapter 3.6].

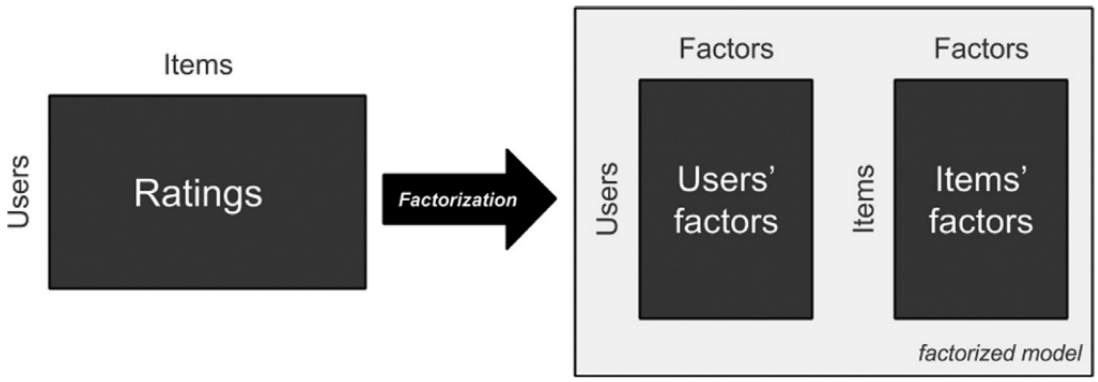


Figure 5: Simple example of matrix factorization

The m x n matrix R which contains all the ratings of the user to the items can be approximately factorized into two different matrices, matrix U with dimensions m x k and matrix V with dimensions n x k, where k is the number of latent factors. V^T is the transposed version of V. This can be expressed as followed,

$$R \approx UV^T. \tag{2}$$

The rating of user i on the item j can be approximated by the following equation,

$$r_{ij} \approx \overline{u_i} \cdot \overline{v_j}, \tag{3}$$

where $\overline{u_i} = (u_{i1} \dots u_{ik})$ is the latent factor with k different concepts which is the affinity of user i to the different concepts 1 to k. $\overline{v_j} = (v_{j1} \dots v_{jk})$ is the latent factor for the items with the affinity of item j to the different concepts 1 to k. The equation above can be expressed as follows,

$$r_{ij} \approx \Sigma_{x=1}^{k} u_{ix} \cdot v_{jx}, \tag{4}$$

which can be interpreted as the affinity of user i on the concept x times the affinity of item j on the concept x. In many recommendation engines the user-item matrix is sparse and missing some of the values so we can instead use a subset with only observed values from R, denoted O. By doing this all the values in O can also be predicted like this,

$$\hat{r}_{ij} = \Sigma_{x=1}^{k} u_{ix} \cdot v_{jx}, \tag{5}$$

so $\hat{r}_{ij}$ is the predicted value and the r_{ij} is the actual value. The error of a specific entry (i,j) is given by the following expression,

$$e_{ij} = (r_{ij} - \hat{r}_{ij}) = (r_{ij} - \Sigma_{x=1}^{k} u_{ix} \cdot v_{jx}). \tag{6}$$

To minimize the error a gradient descent method can be used. Let J be the sum of all the squared errors. Then we want to minimize the sum of all the errors which leads to the following optimization problem,

$$\min_{U,V} J(U,V) = \sum_{i,j \in O} (r_{ij} - \sum_{x=1}^{k} u_{ix} v_{jx})^2 \tag{7}$$

Stochastic gradient descent is one of the methods which can be used to update the latent factors [3, Chapter 3.6.4.1]. The update can be accomplished with the help of the errors in an individual observed entry. Given an observed entry (i,j) the update equation can be stochastically approximated like this,

$$u_{iq} \leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq} \quad \forall q \in \{1 \dots k\}, \tag{8}$$

$$v_{jq} \leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq} \quad \forall q \in \{1 \dots k\}, \tag{9}$$

as proved by [3, Chapter 3.6.4.1].

Where e_{ij} is the error of a given observed rating, this is used to update the k entries of the row j of V and the k entries of the row i of U. This can also be done simultaneously in vectorized form for better efficiency. This is done until the error has reached an acceptable low value. The α in the equations above is the learning rate.

Because the data usually is very sparse the tendency for overfitting is high. To reduce this regularization can be Incorporated [3, Chapter 3.6.4.2]. The idea of regularization

is to encourage stability by penalizing large coefficients in the U and V matrices and encourage simpler solutions. The regularization term $\frac{\lambda}{2}(||U||^2 + ||V||^2)$, where λ is the regularization constant which regulates how much of the regularization should be used. The regularization term is added to the minimization equation, see Equation (7). By adding regularization the minimization equation will look like the following,

$$\min_{U,V} J(U,V) = \sum_{i,j \in O} (r_{ij} - \sum_{x=1}^{k} u_{ix} v_{jx})^2 + \frac{\lambda}{2}(\Sigma_{i=1}^{m} \Sigma_{x=1}^{k} u_{ix}^2 + \Sigma_{j=1}^{n} \Sigma_{x=1}^{k} v_{jx}^2), \quad (10)$$

as proved by [3, Chapter 3.6.4.2]. With this in mind the update equation will be modifide and look like the following,

$$u_{iq} \leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq}) \quad \forall q \in \{1 \dots k\}, \quad (11)$$

$$v_{jq} \leftarrow v_{jq} + \alpha(e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq}) \quad \forall q \in \{1 \dots k\}, \quad (12)$$

as proved by [3, Chapter 3.6.4.2].

Where the λ is the regularization value used. This value has to be tested to find what fits the algorithm.

To improve the recommendation algorithm we can use user and item biases. The main idea behind this is that the variable holding the users bias will learn, if the user rates very generously or often rates items low. The same goes for the item if the item is, for example, a well-known book will have high bias and if the book has got many bad reviews it will have a lower bias. The user bias p_i starts as a vector with random small values and have the length of the number of user. The item bias q_j also starts as a vector with small random values and the length of the number of users. By including the biases the prediction Equation (5) will be modified. It will now look like this,

$$\hat{r}_{ij} = \omega + p_i + q_j + \Sigma_{x=1}^{k} u_{ix} \cdot v_{jx}, \quad (13)$$

where p_i is the bias of user i, q_j is the bias of item j and ω is the global rating average. This will change the error Equation (6) in the following way,

$$e_{ij} = r_{ij} - \hat{r}_{ij} = r_{ij} - \omega - p_i - q_j - \Sigma_{x=1}^{k} u_{ix} \cdot v_{jx}. \quad (14)$$

One way of incorporating the bias into the algorithms is to add the user and the item biases to the U and V matrices which are used in the book. Another way would be to update the specific user biases and the specific item biases at the same time as the update for the user factors and the item factors. In this case, the update will look like the following,

$$p_i \leftarrow p_i + \alpha(e_{ij} - \lambda \cdot p_i), \tag{15}$$

$$q_j \leftarrow q_j + \alpha(e_{ij} - \lambda \cdot q_j). \tag{16}$$

Book store Example
Imagine having to browse all the different books a book store has, that would take ages. This is where a recommender system can help the buyer. Assume that the user Andreas have read the following books ”Sapiens” (book id 0), ”Blue moon” (book id 1) and ”A Brief History of Time: From the Big Bang to Black Holes” (book id 2) and want to find a new book to read. The store has two books which Andreas has not read, those are ”Elon Musk: Tesla, SpaceX, and the Quest for a Fantastic Future” (book number 3) and the last book is ”The Last Wish” (book number 4). In the book store database it is stored that the user Emilia have read the books number 1, 2, and 5, and the user Jakob which have read the books number 0 and 3.

If we use the matrix factorization method explained in Section 3.2.2 the user-item matrix would look like the matrix in Figure 6

	book 0	book 1	book 2	book 3	book 4
Andreas	4	2	5		
Emilia		4	1		5
Jakob	4			4	

Figure 6: The R matrix

The new step in the algorithm is to initialize random values for the matrices U and V (see Figure 7). The two matrices are then multiplied together to get the whole rating matrix and not only the ratings of the already read books (see Figure 8). The matrices U and V have in this example three latent factors. The number of factors varies depending on the data. In this case we could look at the three different factors like the following, the first one is how much the book is nonfictional. The second one is how much the book is fictional and the last factor is about how much the book belongs in the history genre. The meaning of the latent factors is given by the users' consummation habits and not a static variable.

$$
\begin{array}{c}
\\
\text{Andreas} \\
\text{Emilia} \\
\text{Jakob}
\end{array}
\begin{bmatrix} 1.5 & 0.2 & 2 \\ 0.1 & 0.2 & 0.5 \\ 1.2 & 0.0 & 1.9 \end{bmatrix}
\times
\begin{array}{c}
\begin{array}{ccccc} \text{book 0} & \text{book 1} & \text{book 2} & \text{book 3} & \text{book 4} \end{array} \\
\begin{bmatrix} 1 & 0.2 & 1.9 & 1 & 0.9 \\ 0.2 & 0.2 & 1.1 & 0 & 1.9 \\ 0.1 & 0.5 & 1.8 & 1.2 & 0.9 \end{bmatrix}
\end{array}
$$

Figure 7: Untrained matrices U x V

$$
\begin{array}{c}
\\
\text{Andreas} \\
\text{Emilia} \\
\text{Jakob}
\end{array}
\begin{array}{c}
\begin{array}{ccccc} \text{book 0} & \text{book 1} & \text{book 2} & \text{book 3} & \text{book 4} \end{array} \\
\begin{bmatrix} 1.74 & 1.34 & 6.67 & 3.9 & 3.53 \\ 0.19 & 0.31 & 1.31 & 0.7 & 0.92 \\ 1.39 & 1.19 & 5.7 & 3.48 & 2.79 \end{bmatrix}
\end{array}
$$

Figure 8: The R matrix approximated by the untrained matrices U and V

The result of the untrained U and V matrices was not a success (see Figure 8). To improve this, an error equation and the stochastic gradient descent are introduced to minimize the error. The error can only be calculated from the book which a user has read so the error equation will be, error = real rating by the user - the approximated rating. The error equation can be calculated for the user Andreas on the books 0, 1, 2, the user Emilia on the books 1, 2, 4, and lastly on the user Jakob on the books 0 and 3. To calculate the general error for all users root mean squared error, see Equation (19). This yields the total root means square error of: 6.76558. The goal is to get a low error value. After a bit more of training and adjusting the latent factors, the matrices in Figure 9 was created, in the left matrix we can interpret that Andreas likes nonfiction books, dislike fiction books and quite like history. In the right matrix we can interpret that book number three would suit Andreas the most, because the book in a nonfiction book with little fiction in it, but it does not contain any history which is a shame for Andreas. Book number four has very much fiction which Andreas dislikes but instead has some history in it, but not enough to make it a good recommendation.

The two trained matrices U and V are then multiplied which gives the result seen in Figure 10. That matrix has a total root mean square error of 2.3 which is an improvement and with further training the root mean square error will be even lower. Now given the complete matrix, given by the matrix multiplication of the trained matrices V and U the complete matrix can be used to recommend new books to the users. So for Andreas, book number three would be the best to recommend, for Emilia none of the books would suit her which could be a warning for the store to buy in some new books that suits Emilia. For Jakob book number two would be recommended.

$$
\begin{array}{c}
\\
\text{Andreas} \\
\text{Emilia} \\
\text{Jakob}
\end{array}
\begin{bmatrix} 1.7 & 0.3 & 1.4 \\ 0.2 & 1.8 & 0.5 \\ 1.4 & 0.4 & 1.2 \end{bmatrix}
\times
\begin{array}{c}
\begin{array}{ccccc} \text{book 0} & \text{book 1} & \text{book 2} & \text{book 3} & \text{book 4} \end{array} \\
\begin{bmatrix} 1.8 & 0.2 & 1.9 & 1.5 & 0.1 \\ 0.2 & 1.5 & 0.1 & 0.5 & 1.9 \\ 1.2 & 0.5 & 1 & 0.2 & 1 \end{bmatrix}
\end{array}
$$

Figure 9: Trained matrices U x V

$$
\begin{array}{c}
\\
\text{Andreas} \\
\text{Emilia} \\
\text{Jakob}
\end{array}
\begin{array}{c}
\begin{array}{ccccc} \text{book 0} & \text{book 1} & \text{book 2} & \text{book 3} & \text{book 4} \end{array} \\
\begin{bmatrix} 4.8 & 1.49 & 4.66 & 2.98 & 2.14 \\ 1.32 & 2.99 & 1.06 & 1.3 & 3.94 \\ 4.04 & 1.48 & 3.9 & 2.54 & 2.1 \end{bmatrix}
\end{array}
$$

Figure 10: The R matrix approximated by the trained matrices U and V

3.3 Content-based filtering

A CBF recommender system relies on features from the user and the items. This system recommends items with almost the same types of features as the user has liked previously [12]. One example of this could be a simple movie recommendation system, where all the movies have a genre, cast, director which is the item features. When a user watches a movie and ranks it high the recommender system will recommend similar movies with the same genres and/or similar cast and so on. By doing this a user profile is created with the user's preferences and taste, which will be used by the recommendation system to recommend more personalized recommendations of movies to the user [12]. CBF recommendations are to prefer when a new item is added to the system. The new item has a higher chance of being recommended to users because the algorithm doesn't take into account how many users have rated the item but instead only looks at the metadata about the item and makes recommendations based on that [3, Chapter 4.1]. One drawback of the CBF approach is that the system can get very specialized and only suggest similar items and not something unexpected which can be appreciated [6].

The core principles of CBF systems are explained in the book by Aggarwal [3, Chapter 4.2], and are narrated in the rest of this section. CBF methods uses two kinds of data to function, the first is the item features, an example could be a text description of the item or a category. The second kind of data is a user profile where the feedback from the user's behavior is registered. This could be the user's rating on a specific item or an implicit action like clicking on an item. By collecting this kind of data the system

has information about the user's preferences and also information about the items. CBF is commonly used in systems that have a high amount of information about the items. Recommendations on web-pages are a common example of CBF because the pages have a lot of text and usually unstructured. This method can be divided into three different stages, a preprocessing portion, a learning phase, and lastly a prediction phase. The two first steps are done offline where a classification or regression model is created and trained. The last step is done online to produce the recommendations to the users. In the preprocessing and feature extraction phase, the different important features are converted into keyword-based vector-spaced representation. Then the user profile is created based on the users previously activity in the system, like leaving feedback on an item in the form of rating or consuming/buying. This information is used with the item's features to produce the training data which the classification or regression model uses to train on. The trained model is also referred to as the user profile. In the final step the user profile is used to make recommendations.

In the article [13] Philip et al., produced a system that recommends research papers to the user based on the users taste and the description of the items. It is stated in the article that in systems where the number of items is more than the number of users, a CF method is ineffective, that's why the authors used a CBF approach to make the recommendations. The technique they used was the term frequency-inverse document frequency (TF-IDF) and cosine similarity. The main idea by using TF-IDF and cosine similarity is to first find which words in a text that are important and then looking for similar text documents with the same important words according to Neto et al., [14]. $\mathrm{TF}(w, d)$ represents the term frequency of the word w in the document d, the higher the $\mathrm{TF}(w, d)$ is the more representative is that word for the document. The $\mathrm{DF}(w)$, document frequency, represents how many different documents the word w occurs in. The inverse document frequency which is the inverse of document frequency means that if $\mathrm{IDF}(w)$ for w is high it means that the word w only occurs in a few selections of documents while if $\mathrm{IDF}(w)$ is low it means that the word w occurs in many different documents and are not descriptive for a specific document [14]. Putting these two expressions together $\text{TF-IDF}(w, d)$ is obtained, which have the equation,

$$\text{TF-IDF}(w, d) = \mathrm{TF}(w, d) \times \mathrm{IDF}(w) = \mathrm{TF}(w, d) \times (1 + log(\frac{|D|}{\mathrm{DF}(w)})), \qquad (17)$$

as provided by [14]. Where w is the word, $d \in D$ where d is a single document and D is all the documents.

When the TF-IDF has been calculated for all the given documents a vector representation of the documents is given. These representations can be used to calculate the

similarities between the documents using cosine similarities, see Equation (1). With the adjustments that x and y are the weights of two words in two different documents. Given a document a similar document can be recommended which includes the same keywords. This can also be done for any items, given a text-based description of the item another set of items with the same keywords in their descriptions can be recommended to the user. Storing these keywords for the items and storing which keywords the user likes recommendations can be improved over time.

3.4 Hybrid recommender system

A hybrid recommender system is a recommender system that merges CBF and CF into one recommender system. By merging the two different types of recommender systems they complement each other's drawbacks. When a new user starts using the service, the CBF recommender system can have a more important part which decreases the effect of the cold-start problem discussed in Section 3.2. When the user has rated some items in the service, the CF system can make a recommendation from a more varied item pool. This decreases the effect of the specializing problem discussed in Section 3.3. Different types of collaborative filtration can also be used which also classifies as a hybrid recommender system [3, Chapter 6.1]. The implementation of a hybrid system can vary, one can use weights on all different recommendation algorithms which will result in some algorithms dominating in some cases and be submission in other cases. One can switch between different algorithms, this can be used when a new user enters the system. If the recommender system contains one CF algorithm and one CBF algorithm the system can prioritize the CBF for the new user and CF for all the other users. There exist more types of hybrid systems for example cascade systems, feature augmented systems, and so on [3, Chapter 6.1].

3.5 Evaluation types

The evaluation of a recommender system can either be done by using offline methods or online methods. Online methods require active user participation because in these methods the user's reaction to the recommendations is measured. One important method of offline evaluation is to calculate accuracy. This can be done either by predicting a rating of an item as precise as possible, mean squared error, and mean absolute error are commonly used when calculating error for precision accuracy. The other way of using accuracy is by ranking items, where the receiver operating characteristic curve can be used to evaluate the ranking aspect, precision and recall can also be used [3, Chapter 7.1]. One way of collecting the rankings by the user is through explicit feedback. This

can be done by prompting the user after the user have finished consuming the item. Despite requiring more from each user this strategy provides more accurate information than only focusing on the implicit feedback. By asking the user about their opinion on a certain object the recommender system becomes more transparent [1].

The most often used metric to evaluate recommender systems is the accuracy. This is because it does not use any real-world users and can be benchmarked against an already tested algorithm. By only using accuracy the user experience is not fully evaluated. Many secondary goals are also important to evaluate like novelty, coverage, trust, and serendipity. Some of the metrics are subjective and to measure some of them numerically can be a challenge because they lack hard measurements [3, Chapter 7.1].

3.5.1 Online metrics

There are two types of online metric, the first on is user studies and the second one is online evaluation. The difference is how the users are recruited to be part of the studies. In the user studies the users are actively recruited to interact with the recommendation system while the interactions are collected and monitored. The users can be asked to do a specific task and then all the interactions can be stored and the feedback from the users can also be stored to then adjust the recommendation algorithm in a way that enhances the experience. By using user studies the changes in various aspects like the algorithm or the user-interface can be monitored and adjusted for the better. One downside is that all the users are aware that they are being monitored and all their interactions will be stored and analyzed. This can lead to some bias in the choices of the user. Another downside is that the recruited users do not represent the whole user population and the recruitment process itself may be biased, because some users are not willing to participate and the users which are may have an interest in the subject [3, Chapter 7.2.1]. For example, if the recruitment is for ranking movies, people who are interested in movies will be more encouraged to participate. Lastly it is hard and expensive to find users who are willing to evaluate the recommender systems [3, Chapter 7.2.1].

The second type of online metrics is online evaluations which also use users, but unlike user studies, online evaluations usually utilize the end-users which work and operate the systems as intended. Because of this the users' bias are usually much lower. One popular metric to use in online evaluation is the conversion rate, which means the frequency that a user chooses a recommended item. Measuring the impact of different recommender systems on the end users can be called A/B testing. The idea is that the users are divided into two groups, group A and group B. Group A interacts with recommender algorithm A and group B interacts with recommender algorithm B. All the other conditions are kept the same for the two groups. This is done for some time and at

the end of the experiment the conversion rate or other evaluation metric are compared between the two groups. One disadvantage is that a large number of users are needed to make a realistic result. Another point is that this method can only be used by the owner of the product. Because of this it is hard to make it generalized for benchmarks [3, Chapter 7.2.2].

3.5.2 Offline metrics

Offline metric if one of the most popular methods of evaluating a recommender system because of the standardized frameworks and statistical robustness [3, Chapter 7.2.3]. Offline metrics use historical data which can be ratings of a user on a specific item. One disadvantage is that this type of method uses historical data of a user which may not reflect how the user will respond in the future. By using offline methods serendipity and novelty cannot be measured which can have a long-term effect on the recommender algorithm [3, Chapter 7.2.3]. The authors of [3, Chapter 7.2.3] continue by stating that one of the popular measurements is accuracy which can be calculated by taking the actual rating of a user on an item and subtract that from the predicted rating. This is done for a subset of all the actual ratings and averaging by using absolute values or squared values to get an overall error. This type of evaluation can be done for systems with explicit ratings but some systems only use implicit ratings, then a type of top-k ranking can be used to evaluate the recommender system.

To calculate the accuracy of rating predictions the rating data is split into two different subsets, one for training and one for evaluating. The system predicts the rating of the evaluation data and compares it to the actual rating. To calculate the average error of the prediction, the mean squared error and the root mean squared error can be used which is defined as follows,

$$\mathrm{MSE} = \frac{\Sigma_{i,j \in E} e_{ij}^2}{|E|}, \tag{18}$$

$$\mathrm{RMSE} = \sqrt{\frac{\Sigma_{i,j \in E} e_{ij}^2}{|E|}}, \tag{19}$$

defined in [3, Chapter 7.3.1, Functions 7.1 and 7.2] Where e_{ij} is the error between the real rating and the predicted rating, i is the user id, j is the item id and E is the set of all user-item ratings used in evaluation.

The recommended items can also contain an important value, this can be used if an item is very profitable for a company, and the company may want to recommend the

item more often. It can also be used to recommend items which are not so obvious [3, Chapter 7.2.1]. Take an online store, for example, a recommendation may be very accurate and similar to the bought item but the user would eventually buy the item anyway, then the recommendation was accurate but not so effective.

Another way of evaluating a recommender system is to look at how many of the recommended items which will be consumed by the user [1, 3]. Take for example Viaplay, if Viaplay recommends the user a set of recommendations, then the user might eventually consume a subset of the recommendations. The more of the recommended items that the user consumes the better the recommender system is at guessing what the user wants. The recommendation algorithm can recommend any number of items to the user but which of the recommended items are relevant? If the number of items recommended to the user is too small the recommended list will miss relevant items, but if the list is too long the relevant items will be disguised by items that are not as relevant. To get the right size of the recommended list a trade-off between false-negatives (when the list is too short and misses relevant items) and false-positives (when the list is too long and items which are not as relevant is displayed) has to be made [3, Chapter 7.5.4]. The trade-off can be quantified using curves with the help of different metrics and two examples of this is the receiver operating characteristic (ROC) curve and the precision-recall curve.

Given the top-k of a recommended list for a user and denote the set of recommendations as $S(k)$. Then the $|S(k)| = k$, when k change so does the size of $S(k)$. G denotes the set of relevant items that the user has consumed. Given this assumption precision is defined as follows "the precision is defined as the percentage of recommended items that truly turn out to be relevant (i.e., consumed by the user)." [3, Chapter 7.5.4]. Aggarwal, the author of the book [3] also states the following about recall, "The recall is correspondingly defined as the percentage of ground-truth positives that have been recommended as positive for a list of size t". Where the ground-truth the true set of relevant items. The author continues with defining the equations as following,

$$\text{Precision}(k) = 100 \cdot \frac{|S(k) \cap G|}{|S(k)|}, \tag{20}$$

$$\text{Recall}(k) = \frac{|S(k) \cap G|}{|G|}. \tag{21}$$

Another measurement stated in the book is the F_1-measurement which uses both preci-

sion and recall and is the harmonic mean between the two and is defined like this,

$$\mathrm{F}_1(k) = \frac{2 \cdot \mathrm{Precision}(k) \cdot \mathrm{Recall}(k)}{\mathrm{Precision}(k) + \mathrm{Recall}(k)}. \tag{22}$$

Isnikaye et al., [1] simplify the equations by saying that the precision is the fraction of actual relevant items to the user of the recommendations and the recall is the fraction of actual relevant items which is a set of the recommended items. These equations can be summarized as follows,

$$\mathrm{Precision} = \frac{\text{Correctly recommended items}}{\text{Total recommended items}}, \tag{23}$$

$$\mathrm{Recall} = \frac{\text{Correctly recommended items}}{\text{Total useful recommended items}}. \tag{24}$$

These equations are from the report by Isnikaye et al., [1].

Recall and precision works for one user on one specific length, k, of recommendations. To make a more general evaluation mean average precision (MAP) and average recall (AR) can be used. To calculate the MAP, the precision for all evaluated users in a specific k is calculated and then averaged. The mean values of all the averaged precision is then calculated which results in the mean average precision [15]. A simple example of how MAP is calculated is presented in the Figure 11 which was inspired by [16] where k is equal to 10. The average precision is calculated for two different recommendations list with both a length of 10, but for two different users. After that is is averaged again which results in the mean average precision. To calculate the AR the highest recall value obtained for the length k is averaged for all the users.

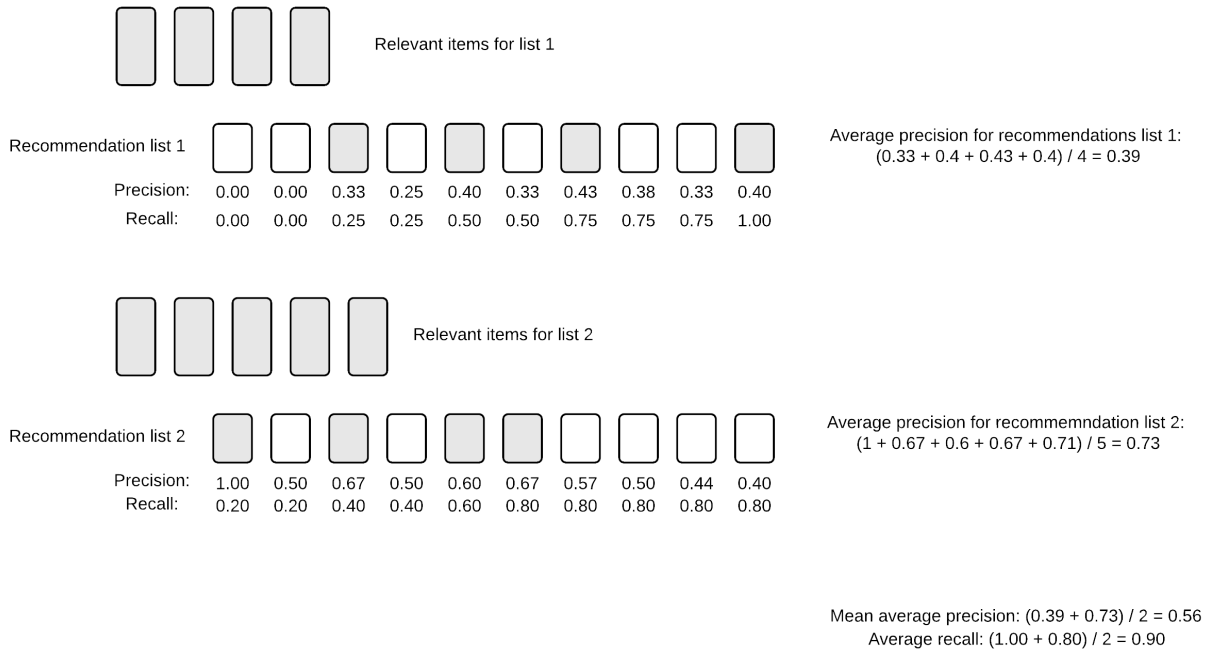


Figure 11: An example of how to calculate the mean average precision

4 Implementation

In the implementation section the knowledge acquired in the theory section has been used to produce the recommender system. First the data of the application will be described which is the building foundation of a recommender system. Following that a description of how the data was used to produce recommendations to the user and what kind of algorithms which was used in what cases. Lastly an idea of how the front-end of the recommendations could be presented to the user is discussed.

4.1 Application data

The data required by the recommender system to work are the user id, item id, and the ratings of a specific user on a specific item. Beyond that the users and items life goals are needed. In the data provided by Tigerhall the total number of users is 15220 and the total number of items is 522 (eBooks: 101, event: 77, podcasts: 344). The total amount of ratings done by users is 13217. The sparseness of data can be calculated by ”total number of actual ratings” / ”total number of possible ratings”. This means that the sparseness of the data is $\frac{13217}{15220 \times 522} = 0.00166359549$.

At the beginning of the project Tigerhall stored information about what a user has consumed and not any information about the ratings on the item. A rating system was planned to be implemented at the beginning of the project but due to some unforeseen events the implementation was delayed. Due to this the ratings were implemented as displayed below:

Rating 0 - The user has not seen the item.

Rating 1 - The user has first bookmarked the item and then removed the bookmark.

Rating 2 - The user has bookmarked the item.

Rating 3 - The user has started consuming the item.

Rating 4 - The user has finished consuming the item.

Both the K-nearest neighbors (KNN) and the matrix factorization algorithm use these ratings to make the recommendations. Two algorithms that usually use explicit data are now using modified implicit data to determine the recommendations.

In the application data only around 80 items have been interacted with, which is presented in Figure 12, where the items have been sorted so the item number does not correspond to the actual item number. The rest of all the 522 items are unseen items.

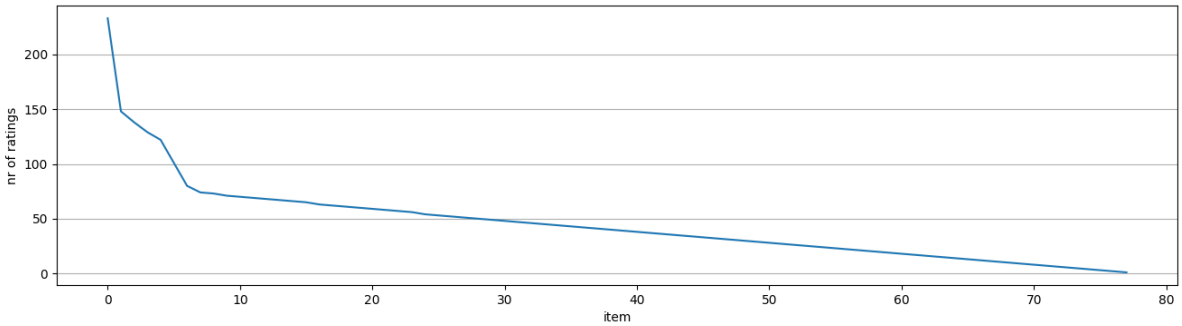


Figure 12: The number of ratings in presented on the y-axis and the item number is presented on the x-axis

4.2 Recommender system

The recommender algorithm is an add-on to Tigerhall's application. The system recommends items to a specific user depending on two things. The first thing is, when a new user starts using Tigerhall's application the user has to select one to three different "life goals". These life goals are like genres that the user is interested in. It can be "I want to be a better leader" or "I want to make more money", depending on the selected life goal the user will have specific recommendations. The second thing the algorithm looks at is how much the user has previously consumed. If it is a new user to the service the user will not have consumed anything and the life goal will determine what will be recommended, but as the user consumes more items the recommendations will partly depend on the life goal and partly depend on other users' consumption. The latter one is done with either k-nearest-neighbor (see Section 3.2.1) or matrix factorization (see Section 3.2.2).

When the mobile application wants to display the dashboard or get recommendations after the user has finished an item, the mobile application which has a Golang (Go) client will send a request to the recommender system which is on a Python server. The request will go from the client to the server which will then run the main recommendations algorithm, see Algorithm 1. This algorithm requires information about the user's consumption and information about the items. To receive this information the algorithm will make a request to the postgreSQL database to get information about the specific

user and all the items. This information is used to determine what to recommend. Depending on how much the user has previously consumed the recommender algorithm will use different algorithms to determine which items that should be recommended. This is an example of a hybrid recommender system, see Section 3.4 for more information about hybrid systems.

Algorithm 1: Recommendation

```
1:  function RECOMMEND(user, i)          ▷ Where user - the user which want
    recommendations, i - how many recommendations
2:      recommendations = []
3:      count = user_consumed_count()
4:      if count == 0 then
5:          items = life_goal.recommend()
6:                  ▷ Recommend the most popular items from the user's life goal
7:          recommendations.append(items)
8:      end if
9:      if count > 0  count < 50 then
10:         items = KNN.recommend()
11:                          ▷ Recommend items based on the KNN Algorithm 2
12:         recommendations.append(items)
13:     end if
14:     if count >= 50 then
15:         items = MF.recommend()
16:             ▷ Recommend items based on the matrix factorization algorithm 3
17:         recommendations.append(items)
18:     end if
19:     recommendations.sort()              ▷ Sort the items in descending order
20:     return recommendations[:i]          ▷ Only return the top i number of items
21: end function
```

When the algorithm is finished a list of recommendations will be sent to the client which will be displayed on the mobile application. Figure 13 is a representation of the interactions between the mobile application, client, server, and the database.

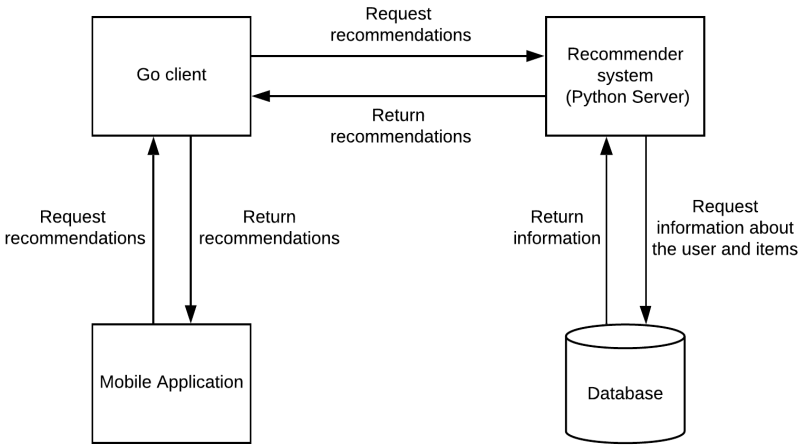


Figure 13: An overview of the system

If the user has not consumed any items then the recommender algorithm will recommend the most popular items from the user's life goal. If the user has instead consumed at least one item and at maximum 50 items the recommender algorithm will use the KNN algorithm, see Algorithm 2. In this case, similar users to the target user are acquired. From these users the items with the highest rating that the target user has not consumed will be recommended. See Section 3.2.1 for a more detailed explanation about KNN.

Lastly, if the user has consumed 50 or more items the recommendation algorithm will use the matrix factorization algorithm, see Algorithm 3. To be able you use the matrix factorization algorithm the latent factors have to be trained. This is done by the Algorithm 4, which describes how the latent vectors are trained. This has to be done before a recommendation can be done. The training algorithm has been modified to fit this project but the base algorithm can be found in the book [3, Chapter 3, Figure 3.9]. When the latent factors have been trained, ratings of any item can be calculated using the latent factors. By ordering all the ratings on all the items the most suitable items can be recommended.

The recommender algorithm 1 can be changed so that it return recommendations from all the algorithms and not only one. This will be the case when the user has consumed at least 10 items and will get recommendations based either on other users or based on the user's consumption history.

4.3 Mobile application

The idea is to display the recommendations in a similar fashion as Netflix (see Figure 1a). Now the application has the life goal as header and all the items in a horizontal scroll. In the Figure 14 the header ”Influence People” is displayed which is a life goal and under the header, different types of items with the life goal ”Influence People” is displayed. In this example one book and one podcast is shown and if the user swipes to the left more items will be displayed. In the final product other headers will be displayed, for example ”Based on your consumption history” and all the recommendations from the matrix factorization will be displayed in an ordered manner. Another header could be ”Based on similar users” and here all the KNN items will be displayed. Lastly a header could be ”Recommendations for user [username]” here a mix of the recommendations from all the algorithms would be displayed.

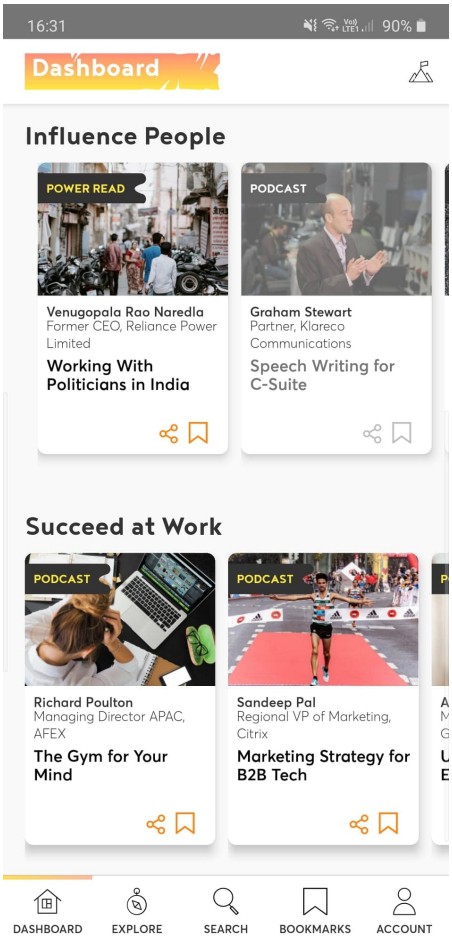


Figure 14: An example of the dashboard of the Tigerhall application

Now a simple example of how a user can interact with the application will be described. Julia is a user who is new to the system. When Julia first log in to the application she will be greeted with different life goals, she has to select one to three different life goals. Let us say she choose ”Influence People” and ”Succeed at Work”. Then her dashboard will look like the Figure 14. The headers will be her life goal and the items shown first will be the most popular item in that life goal. When Julia has started consuming some items the next time Julia logs in to the dashboard it will look different because the KNN algorithm have been used and the matrix factorization has trained it’s latent factors with the new information about Julia. Because of this new headers will be displayed. The new header could be ”Based on similar users” or ”Based on your history”. In these rows the recommendations will be ordered by the prediction score of the unseen items. Because of the low amount of items per life goal one huge benefit with displaying items from multiple life goals in the same row is that if a user consumes all the items in one life goal the user will not be left with no items. It also increases the diversity of the recommendations. The user may also find the new items interesting which will lead to the user also wanting to consume items in the new life goal. This leads to a higher serendipity.

5 Results

To evaluate the recommender system accuracy, MAP and AR are used. All these methods have been presented in Section 3.5. The dataset was divided into two different subsets, a training set and a testing set with the split 90%/10%. The accuracy of the matrix factorization method was done by calculating the root mean squared error for all the user-item ratings in the testing set. A prediction of all the ratings was done by the matrix factorization method then compared with the actual ratings. The accuracy in the form of root mean squared error could only be done for the algorithm which uses ratings. The KNN method does not use user-item ratings but instead recommends a list of items that should fit the user based on similar users' consumption. To evaluate this algorithm MAP and AR were used.

The training accuracy for the matrix factorization is around 4 when calculating RMSE, the evaluation RMSE varies depending on the number of consumed items the evaluation user have. If the user have consumed 200 items the accuracy is around 4 but if the user have only consumed 1 item the accuracy is up to 24, see Figure 15 for a more detailed presentation.

The AR are presented in the Figures 16 and 17. The MAP is presented in the Figures 18 and 19. The red line is representing the matrix factorization method and the number of latent factors was experimentally decided to 56. With a lower number, the training time was too long to achieve the same amount of accuracy and with a higher number, the accuracy also started to stagnate, probably because it started to overfit and work against the regularization. The blue line is a boosted version of the same matrix factorization. The boosted MF algorithm increases the chance of items that have the same life goal as the user to be recommended. The yellow triangles are the KNN algorithm and the parameters used in the KNN were based on how many recommendations were needed. If the algorithm needed 50 recommendations then the number of the closest users would be at minimum 5 and the number of items taken from each user is 10. If one of the users don't have 10 items consumed which the target user has not consumed another user is taken into an account resulting in 11 users. In this case, the number of users is the value K in K-nearest neighbors. The red circles are a popularity algorithm that recommends the most popular items to all the users, this one and the random algorithm serves as a baseline.

It is clear in all the AR figures that the user-centered KNN algorithm have the highest AR. We can also see that the MF algorithm increases its AR with the user consumption count. We can also see that the boosted MF is slightly better than the regular MF in all the AR cases. All the MAP scores are quite low besides the user-centered KNN algorithm which has the higher MAP score of them all, despite in 18a where the KNN

algorithm is low. This can be due to the low amount of evaluation users, the same goes for the recall with 200 consumed items. We can also see that the user-centered KNN algorithm always have higher AR value and that it also have a higher MAP value than the popularity algorithm which Tigerhall partly uses now. The popularity algorithm just recommendes the most popular items to all the users.

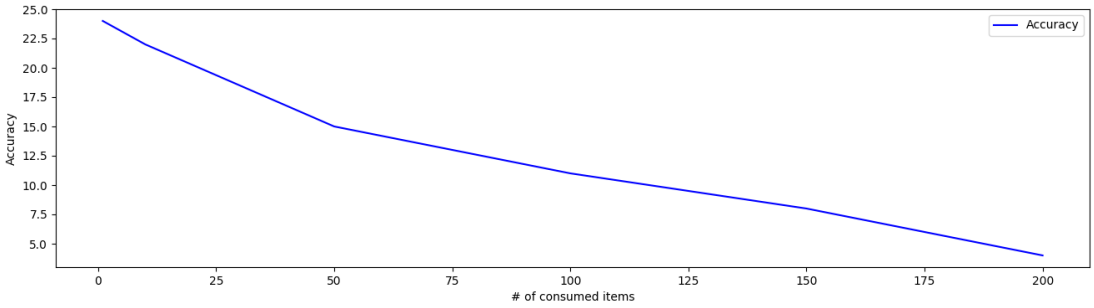
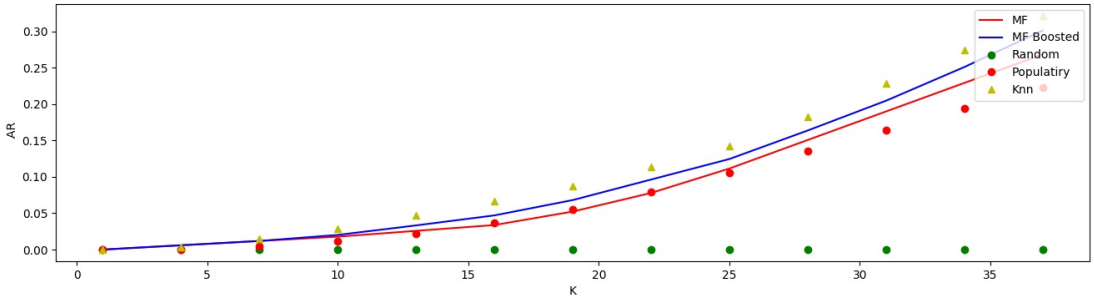
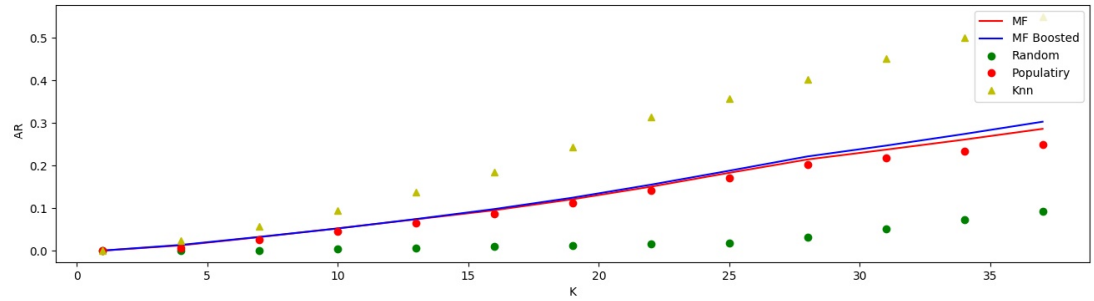


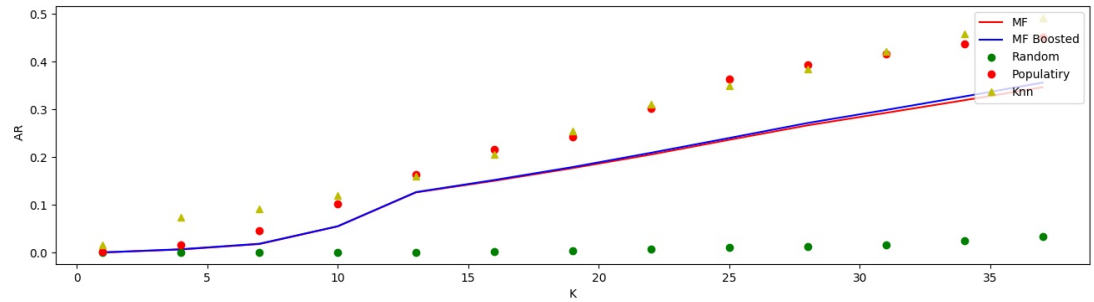
Figure 15: The accuracy of the MF algorithm with the accuracy on the y-axis, and the number of consumed items on the x-axis.



(a) Recall when the number of consumed items are 200 or above.

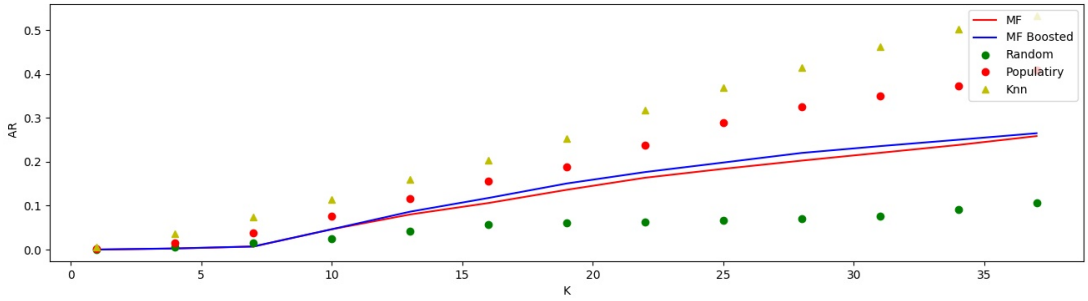


(b) Recall when the number of consumed items are 150 or above.

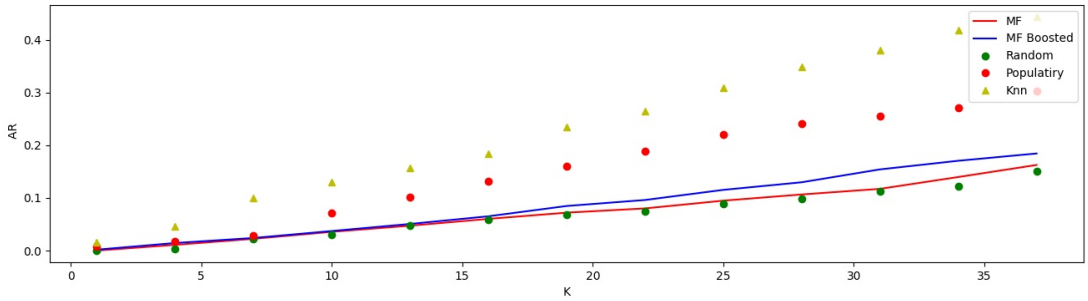


(c) Recall when the number of consumed items are 100 or above.

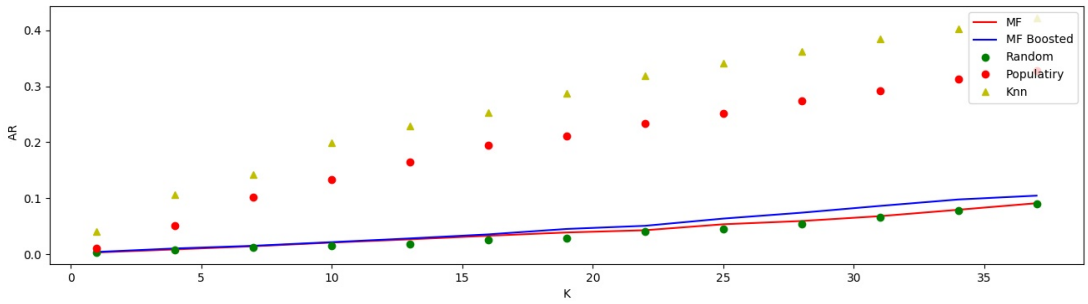
Figure 16: In this figure three different recall graphs are shown. The evaluation users consumption is the key difference where it varies between 200 items down to 100 items.



(a) Recall when the number of consumed items are 50 or above.

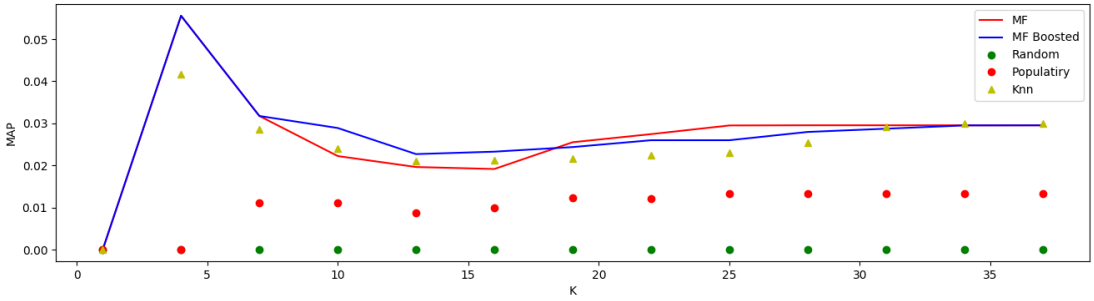


(b) Recall when the number of consumed items are 10 or above.

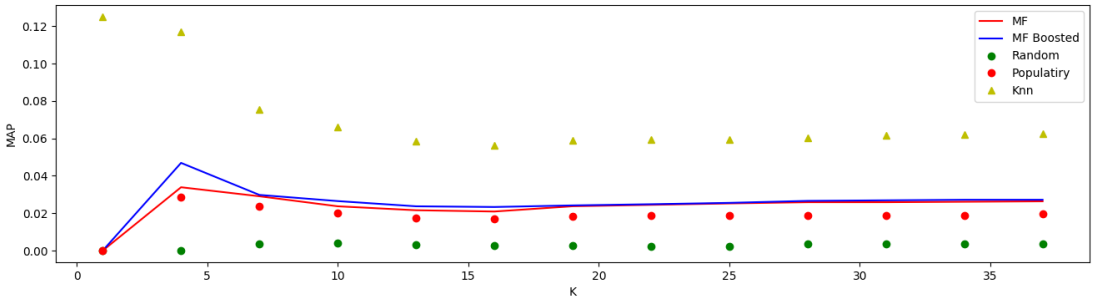


(c) Recall when the number of consumed items are 1 or above.

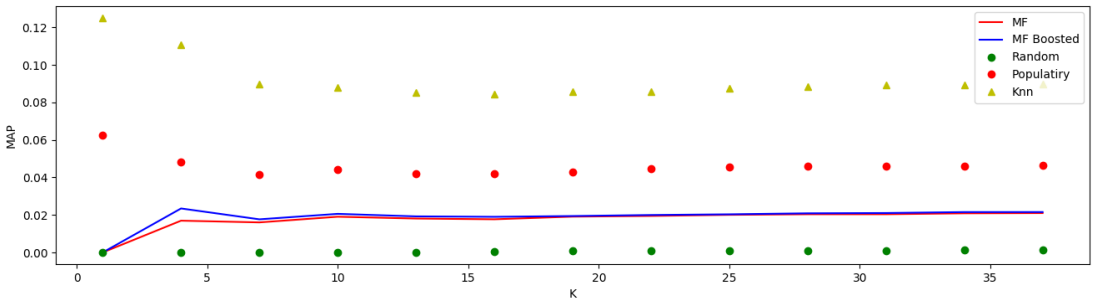
Figure 17: In this figure three different recall graphs are shown. The evaluation users consumption is the key difference where it varies between 50 items down to 1 items.



(a) Precision when the number of consumed items are 200 or above.

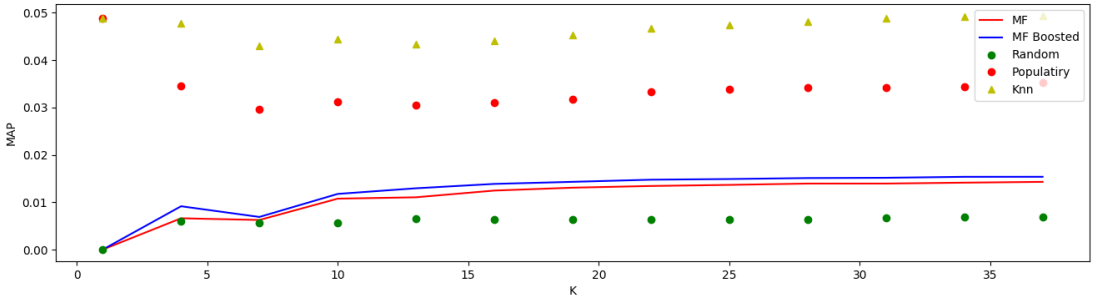


(b) Precision when the number of consumed items are 150 or above.

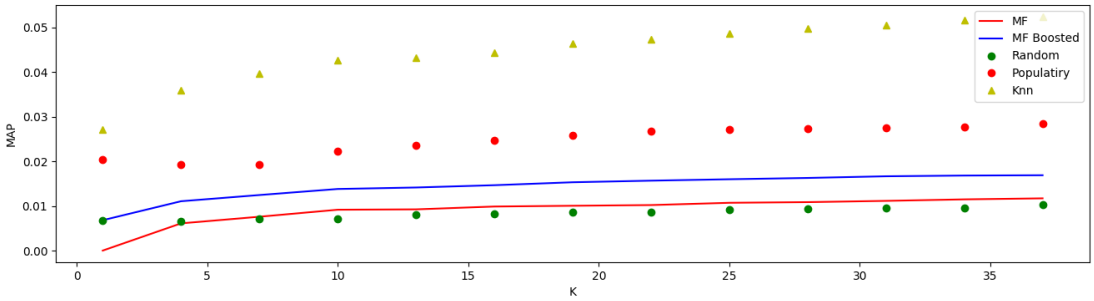


(c) Precision when the number of consumed items are 100 or above.

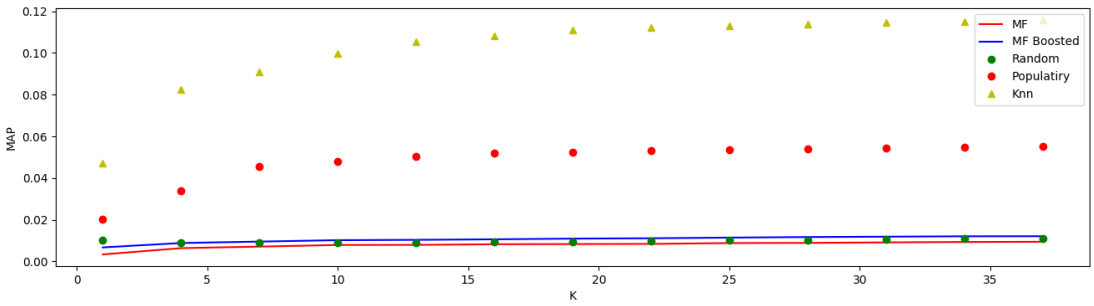
Figure 18: In this figure three different precision graphs are shown. The evaluation users consumption is the key difference where it varies between 50 items down to 1 items.



(a) Precision when the number of consumed items are 50 or above.



(b) Precision when the number of consumed items are 10 or above.



(c) Precision when the number of consumed items are 1 or above.

Figure 19: In this figure three different precision graphs are shown. The evaluation users consumption is the key difference where it varies between 50 items down to 1 items.

The personalization metric is calculated for each of the recommendation algorithms and the result can be seen in the graph 20. A high personalization means that the recommendations received by the users are not alike and differ from each other. The lower the personalization score the more similar the recommendations are. We can see that all the different algorithms have a high personalization score except the popularity al-

gorithms, which have almost zero personalization score. The KNN algorithm is almost as personalized as the random algorithm is.

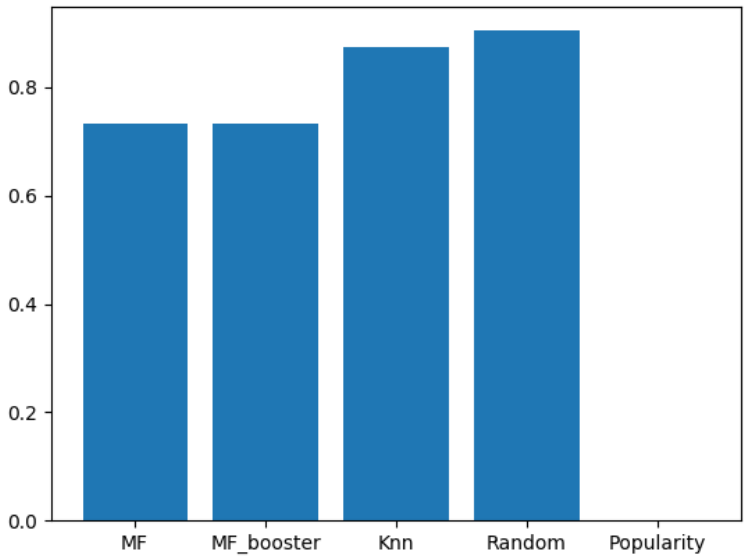


Figure 20: The personalization of all the different algorithms is displayed in this figure with the personalizing score on the y-axis, and the algorithm name on the x-axis.

6 Discussion

When choosing the algorithms for this project the fact that there aren't so many items per life goal was prioritized. This is because if the user consumes every item in a life goal then the user may feel like not using the application anymore. Also, if the recommender system suggest item from another life goal which the user likes then the user may use the application for a longer period because the user has more interesting items to consume. To give the user the feeling of serendipity was also prioritized. To fulfill the latter requirement only a CBF recommender system would not suffice. That is why the main focus was on the CF algorithms. By prioritizing the CF the overspecialization problem, explained in Section 3.3, was handled because it recommends item which is not restricted to a specific life goal. The problem explained in Section 3.2 which was the cold-start problem is handled by using CBF recommender systems for new users which will recommend items with the same life goal as the user. To handle the cold-start for items a new application row will be implemented which will display the new items. Having the new item displayed for the user on the mobile application it gives the item a chance to get some ratings which will increase the chance that it would be recommended by the CF algorithms.

In Tigerhalls data there are approximately 10 life goals and a total amount of 522 items where podcasts are the most popular. Each podcast is about 5-20 minutes long so the user will listen to all the podcasts in a given life goal quite fast. This is a motivation why recommending items that are from another life goal is so important.

The offline evaluation methods can be misleading because of the lack of data and the high sparseness of the data. If an online evaluation of the recommendation algorithms had been done I believe that a more realistic evaluation would have been done. This is because the real world user would be able to explain the feeling of all the recommendations and also to say if the user would consume the recommended item. Another important aspect is that the serendipity could be estimated which in my opinion is a very important evaluation metric because the user can always find similar items to the one which the user has previously consumed but to find an item which is good but unexpected is difficult. In the current version of the application, all the items are divided into rows of life goals. If the user only consumes items with the same life goal then when the items are all consumed (at this point in time the number of items is not high per life goal) the user has to find a new life goal to consume or stop using the application until new items are introduced. To minimize this risk that the user would stop using the application, the recommendations should include items with different life goals that are similar or interesting for the user. If this is done more items can be consumed by the user and the user does not have to make an active choice of choosing a different life goal as well as the previous one. This choice is done by the application and the user will

have a greater diversity of items to choose from.

The poor performance of the matrix factorization can be due to the high sparseness of the data. The algorithm will improve over time because more users will consume more items which will lead to a lower sparseness. This can be endorsed when comparing the different AR graphs, see figure 18 where the more the user has consumed the higher the AR score of the MF algorithms are. With a lower sparseness, there will exist more data about what a specific user likes, with this data a more relevant recommendation can be done. I also think that given a proper rating system where the user will rate the items on a scale between 1-5 stars the matrix factorization will improve. Now the rating system is only looking at if a user has consumed it or not, but what if the user disliked the thing that is consumed? In the current application, the user has no way of telling the recommender system that the item was bad. Instead, the recommender system thinks that all the consumed items are good for the user. The different limits on the switching hybrid recommender system exists because the MF method needs a lot of data about the user's consumption history. If the user has not consumed so many items the MF method will not be able to predict accurately this can be seen in the result section, see 5. In the result, the MF started to show promising results if the number of consumed items exceeds 50, that's why the limit is 50. If the user has consumed over 200 items the MF method was even better but if the limit would be set that high then some user may already have quit using the application because the recommendations are so alike and possibly from the same life goal. The MF method don't take into an account the life goal but instead look at the different item individually. This probably leads to a more diverse recommendation which may keep the user interested in different life goals also. The limits can be changed and the recommender algorithm can also return the results both from the KNN and the MF and display them on separate rows. The KNN row can have the title ”Similar users have also consumed these items” and the MF row can have the title ”Based on your consumption history, these items may be of interest”.

The good performance of the population algorithm, which recommends the most popular items to all the user, can be because that's the way the application partly recommends items now. Given the sparse data the KNN still managed to produce quite good results. To get a more accurate evaluation I would suggest doing an A/B - testing with the different algorithms and then do interviews with some of the users to get a more realistic evaluation and also to capture some of the ”soft” metrics like novelty, serendipity and trust. When doing a A/B - testing the recommended items from the different algorithms would be presented to the user. This will probably increase the consumption rating of the recommended items and result in a higher recall and precision rating.

7 Conclusion

In this project a recommender system has been created which is capable of recommending events, podcasts, and eBooks to a user depending on the user's history. This is also possible with a sparseness of 0.001663, because of the sparseness the matrix factorization was not so successful but the KNN approach was quite good. When the recommender system comes in use the users will be recommended items with different life goals which will give the users a more diverse experience. This is especially true for the MF algorithm which do not rely on the items life goal. By introducing this recommender system in to Tigerhall's application my belief is that the users will find new interesting items to consume which will keep them interested in the application for a longer period.

8 Future work

On major improvement to this project would be to implement a more superior CBF algorithm. This would help the recommender system to improve in cases where the user has not consumed anything and when a new item is introduced. One way the CBF could be implemented is with the use of TF-IDF. This algorithm would take the description of the new item and finding similar items based on the description. By using this algorithm the cold-start problem for items would be smaller and the recommendations would not be restricted to the life goal as it is of today. This would also help Tigerhall to minimize the long-tail problem which they have. Only 80 of 522 items have been seen by the user and with a good CBF filtration algorithm the number of seen items can increase.

To make this project ready for production a more robust evaluation method and a method of how to improve the algorithm over time has to be implemented. Also a front-end to show the recommendations both as a row on the dashboard and also as a popup after a user has finished consuming an item. One thing that would be preferred when deploying the algorithm into production would be to have a rigid test suit. This test suit should test all the core functionality of the algorithm and also all the connections for example that the server can retrieve data from the database. That the client can communicate with the server. That the different algorithms are getting right inputs, etc.

Another thing to keep in mind is that the matrix factorization method has to be retrained approximately once per day to get more relative recommendations. But at run-time this method is fast because it only needs to calculate the dot product of two vectors and some biases. In the meantime the knn method is much slower because it needs to calculate all the distances to all the users and find items that are unseen to the user but still relevant. The distances to all the users can also be stored but it need to be updated regularly because most users consumes new items every day.

References

[1] F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273, 2015.

[2] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6(4), 2015.

[3] Charu C. Aggarwal. *Recommender Systems*, volume 40. Springer, 2016.

[4] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. *RecSys 2016 - Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198, 2016.

[5] Manzar Abbas, Muhammad Usman Riaz, Asad Rauf, Muhammad Taimoor Khan, and Shehzad Khalid. Context-aware Youtube recommender system. *2017 International Conference on Information and Communication Technologies, ICICT 2017*, 2017-December:161–164, 2018.

[6] Ana Belén Barragáns-Martínez, Enrique Costa-Montenegro, Juan C. Burguillo, Marta Rey-López, Fernando A. Mikic-Fonte, and Ana Peleteiro. A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition. *Information Sciences*, 180(22):4290–4311, 2010.

[7] Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. *Proceedings of the National Conference on Artificial Intelligence*, (July):187–192, 2002.

[8] Jens Grivolla, Diego Campo, Miquel Sonsona, Jose Miguel Pulido, and Toni Badia. A hybrid recommender combining user, item and interaction data. *Proceedings - 2014 International Conference on Computational Science and Computational Intelligence, CSCI 2014*, 1:297–301, 2014.

[9] George Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, CIKM '01, page 247–254, New York, NY, USA, 2001. Association for Computing Machinery.

[10] Fernando Ortega, Antonio Hernando, Jesus Bobadilla, and Jeon Hyung Kang. Recommending items to group of users using Matrix Factorization based Collaborative Filtering. *Information Sciences*, 345:313–324, 2016.

[11] Latent variable. https://en.wikipedia.org/wiki/Latent_variable. Accessed: 2020-03-15.

[12] Yashar Deldjoo, Mehdi Elahi, Paolo Cremonesi, Franca Garzotto, Pietro Piazzolla, and Massimo Quadrana. Content-Based Video Recommendation System Based on Stylistic Visual Features. *Journal on Data Semantics*, 5(2):99–113, 2016.

[13] Simon Philip, P Shola, and A Ovye. Application of content-based approach in research paper recommendation system for a digital library. *International Journal of Advanced Computer Science and Applications*, 5(10), 2014.

[14] JL Neto, AD Santos, and CAA Kaestner. Document clustering and text summarization. *Proceedings of the 4th International Conference* {...}, 2000.

[15] Hannah Bast. Information retrieval ws 17/18, lecture 2: Ranking and evaluation. https://www.youtube.com/watch?v=bCVPnnWqY8s&t=4861s, February 2018. Accessed: 2020-05-25.

[16] Victor Lavrenko. Evaluation 12: mean average precision. https://www.youtube.com/watch?v=pM6DJ0ZZee0, January 2014. Accessed: 2020-05-25.

A Algorithms

Algorithm 2: KNN recommendation

```
1: function KNN_RECOMMEND(user)          ▷ Where user - the user which want recommendations
2:     relevant_users = get_similar_users()    ▷ returns the most similar users based on Equation 1
3:     recommendations = relevant_unseen_items(user, relevant_users)    ▷ Returns items per user which the target user have not seen.
4:     recommendations.sort()          ▷ Sort the items with respect to the ratings
5:     return recommendations          ▷ returns the recommendations
6: end function
```

Algorithm 3: MF recommendation

```
1: function MF_RECOMMEND(user)           ▷ Where user - the user which want recommendations
2:     items = get_data_from_db()        ▷ Returns all items in the database
3:     rated_items = get_rated_items(items)    ▷ Return all the rated items
4:     if not latent_factors_trained() then
5:                                       ▷ check if the latent factors have been trained
6:         train_latent_factors()        ▷ Trains the latent factors, see Algorithm 4
7:     end if
8:     for item in rated_items do
9:         prediction = predict(user,item)
10:                ▷ This function uses Equation (13) and only return unseen items
11:        recommendations = recommendations.append(prediction)
12:    end for
13:    recommendations.sort()
14:                                      ▷ Sort the recommendations with respect to the ratings
15: end function
```

Algorithm 4: Training latent vectors

function TRAIN(RM, α, λ, iters) ▷ Where RM - rating matrix with only observed ratings, α - learning rate, λ - regularization rate, iters - number of iterations
    Randomly initialize the user matrix U
    Randomly initialize the item matrix V
    for i in iters **do**
        Shuffle the items in RM
        for item (i,j) in RM **do** ▷ Where i is the user and j is the item
            $e_{ij} = r_{ij} - \omega - p_i - q_j - \Sigma_{x=1}^{k} u_{ix} \cdot v_{jx}$
            $p_i \leftarrow p_i + \alpha(e_{ij} - \lambda \cdot p_i)$
            $q_j \leftarrow q_j + \alpha(e_{ij} - \lambda \cdot q_j)$
            $prev\_u = u$
            for each $q \in \{1 \dots k\}$ **do**
                $u_{iq} \leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq})$
            end for
            for each $q \in \{1 \dots k\}$ **do**
                $v_{jq} \leftarrow v_{jq} + \alpha(e_{ij} \cdot prev\_u_{iq} - \lambda \cdot v_{jq})$
            end for
        end for
    end for
end function