A Matrix Factorisation based Algorithm for Decentralised Personalisation

Clemens Manuel Frank

September 18, 2017

1ST REVIEWER: PROF. DR. STEFAN DECKER
2ND REVIEWER: PROF DR. BERNHARD RUMPE
ADVISOR: DR. BENJAMIN HEITMANN
Acknowledgements

Foremost, I thank my advisor, Dr. BENJAMIN HEITMANN. Without his ongoing support and knowledge, the creating of this document would have not been possible. Always available, he could tell accurately, what to focus on during the work on this document.

Furthermore, I thank my friends Konrad Kollnig, Marc Burian, and Dimitry Sidorov for spell and speech checking, guidance and personal support. To all relatives and friends who in one way or another shared their support – thank you.
Abstract

Nowadays Internet services are personalised by so-called recommender systems, which filter relevant content for a user. When user profiles and content are decentralised in order to enhance privacy, the recommender system has to work distributed as well. In this thesis, we evaluate a distributed recommender algorithm for decentralised Online Social Networks, where preferences of users are only shared between two parties at a time and without a central storage. Our results show that the quality of the recommendations is only 5% worse than recommendations by centralised recommender algorithms, thus the trade-off between accuracy and privacy through distribution is acceptable.
Contents

1 Introduction ........................................... 11
  1.1 Motivation ........................................ 11
  1.2 Thesis Goal ........................................ 11
  1.3 Outline .......................................... 12

2 Background ........................................... 13
  2.1 Online Social Networks ............................. 13
    2.1.1 Privacy Aspects .............................. 13
  2.2 Recommender Systems ............................. 14
    2.2.1 Demographic Filtering ....................... 14
    2.2.2 Content-based Filtering ...................... 14
    2.2.3 Collaborative Filtering ...................... 14
    2.2.4 Hybrid Filtering ............................ 16
    2.2.5 Privacy Aspects ............................ 16

3 Related Work ........................................ 17
  3.1 Distributed Social Networks ....................... 17
  3.2 Distributed Collaborative Filtering ............... 19
    3.2.1 Neighbourhood-based ....................... 19
    3.2.2 Matrix Factorisation ....................... 21
  3.3 Summary .......................................... 22

4 Use Case and Requirements .......................... 25
  4.1 Use Case .......................................... 25
  4.2 Requirements .................................... 25
  4.3 Comparison ..................................... 26

5 Conceptual Approach ................................ 27
  5.1 Distributed Learning Algorithm ................. 27
    5.1.1 Learning rate ................................ 29
    5.1.2 Normalisation ............................... 29
  5.2 Comparison Algorithm ............................ 31
    5.2.1 BiasedMF .................................. 31

6 Implementation ...................................... 33
  6.1 General Infrastructure ........................... 33
  6.2 Code Organisation ................................ 34
List of Algorithms

5.1 Distributed learning algorithm from [IICM11] . . . . . . . . . . . . . . . . . . 29
5.2 Projection of a point onto the canonical simplex of $\mathbb{R}^n$ [Mic86] . . . . . . . 30
1 Introduction

1.1 Motivation

Since the invention of the Internet, the amount of simultaneously available data (e.g. music, movies) has increased drastically. In 2016 the popular online video streaming provider Netflix\(^1\) offered 5,532 movies and TV-Shows\(^2\), the music streaming service Spotify\(^3\) provided more than 30 million songs\(^4\). The crucial question now is, how to help the consumer finding the content he is most interested in. Recommendations, which were formerly done by "word of mouth", are now provided by so-called recommender systems. The recommender system learns about your preferences explicitly (e.g. through ratings on items like movies or songs) or implicitly (e.g. through analysing how often you hear a specific song) and then recommends you new items you may like, based on your preferences. Another field of application for personalisation by recommendation systems are online social networks, in which only user-relevant content is displayed.

However, the fact that the preferences are stored on a centralised server potentially causes privacy issues. Thus there is a trade-off between privacy and quality of the recommendations: The more personal information the server is aware of, the more accurate are the recommendations, but the lower the privacy is and vice versa. Since recommender systems are by now indispensable, privacy enhancing solutions are necessary.

1.2 Thesis Goal

As privacy in common recommender systems with central storage is not guaranteed, the aim of this thesis is to implement a decentralised recommendation algorithm, which deals with privacy threats. Afterwards, the algorithm will be evaluated on a public movie dataset by verification of functionality and comparison to a state-of-the-art centralised recommendation algorithm.

---

\(^1\)Netflix: https://www.netflix.com/
\(^2\)https://www.allflicks.net/netflixs-us-catalog-has-shrunk-by-more-than-2500-titles-in-less-than-2-5-years/
\(^3\)Spotify: https://www.spotify.com/
\(^4\)https://press.spotify.com/us/about/
1 Introduction

1.3 Outline

The remainder of this thesis is organised as follows. Chapter 2 introduces the state-of-the-art centralised online social networks and centralised recommender systems as well as privacy aspects. Chapter 3 deals with related work on the topic distributed social networks and distributed recommender systems. Chapter 4 discusses a use case and requirement list for the implementation. Chapter 5 describes the conceptual approach, while Chapter 6 provides the more specific realisation of the approach. Chapter 7 explains the experiment and presents the results for the implemented algorithm. The thesis finishes with a conclusion and suggestions for further research in Chapter 8.
2 Background

In this Chapter we revise the required background, focusing on two main aspects. Section 2.1 introduces online social networks and related privacy aspects. Section 2.2 describes current recommender systems and their privacy aspects.

2.1 Online Social Networks

Online social networks (OSNs) are very popular on the Internet. The most prominent example, Facebook\(^5\), had 1.94 billion monthly active users in 2017\(^6\) and is thus the third most frequently visited site on the Internet\(^7\). In OSNs users can create a profile describing their characteristics and connect to other users around the world (usually their friends and acquaintances). Furthermore, they can share status updates (e.g. what they are doing right now), photos (e.g. pictures from the holidays), videos etc., and interact with other users' content (e.g. comment on them). Because users tend to share a big quantity of private information with their friends, several privacy issues arise.

2.1.1 Privacy Aspects

Kayes et al. [KI15] classified the privacy issues of OSNs into two main attack categories: Attacks on Users and Attacks on the OSN.

**Attacks on Users** comprises all isolated attacks against a small population of users. Possible initiators of these attacks are other users, when interacting with untrusted users like acquaintances the user barely knows. Other possible initiators are social applications, for example third-party applications with access to a user's profile. Furthermore, the OSN itself is a possible initiator. The OSN stores the personal information of all users and might misuse them for other purposes. In addition, sometimes the OSN publishes anonymised data for research, which can be de-anonymised.

**Attacks on the OSN** cover all attacks targeting the entirety of users, thus the OSN itself. These attacks are for example *Sybil attacks*, which aim to manipulate the public opinion by creating multiple fake identities. Another popular attack is *Distributed Denial-of-Service*

\(^5\)Facebook: https://www.facebook.com/
\(^7\)http://www.alex.com/topsites
Background

(DDoS), which makes a server unavailable by systemically overloading the server’s network traffic. Other attacks are more harmful for the user, like the aggregation of all publicly available user profiles to build databases with the personal information in order to take advantage or sell the gathered data. *Spam* and *malware* attacks on the users of the OSN are also possible.

2.2 Recommender Systems

In general, users provide feedback for a certain group of items (e.g. movies, songs), which the Recommender Systems (RSs) use to learn the preferences of its users to then provide recommendations on items the user will probably like. We refer to explicit user feedback as *ratings*. According to [BOHG13], the mainly used filtering algorithms can be divided into four categories: demographic filtering, content-based filtering, collaborative filtering, and hybrid filtering, which are described in the next paragraphs.

2.2.1 Demographic Filtering

*Demographic filtering* is based on the assumption that users with the same personal attributes (e.g. country, age, gender) will tend to have the same preferences.

2.2.2 Content-based Filtering

*Content-based filtering* analyses the content a user consumes or buys and searches for similar content to recommend. Therefore, the content is characterised by several attributes (or tags), to find similar content.

2.2.3 Collaborative Filtering

The *collaborative filtering* (CF) technique can be basically divided into two different categories:

The *memory-based* methods are based on similarity matrices (e.g. *User-User* or *Item-Item* matrices), which indicate the similarity between two users or items, respectively. A popular similarity measure is Pearson’s correlation coefficient:

\[
    w_{a,u} = \frac{\sum_{i=1}^{m} (r_{a,i} - \bar{r}_a) \cdot (r_{u,i} - \bar{r}_u)}{\sigma_{a} \cdot \sigma_{u}},
\]

where \( w_{a,u} \) denotes the similarity between the users \( a \) and \( u \), \( r_{a,i} \) denotes the rating of user \( a \) for item \( i \), \( \bar{r}_a \) denotes the average rating of user \( a \), and \( \sigma_{a} \) is the standard deviation of user \( a \)’s ratings. Another popular similarity measure is called cosine similarity:

\[
    sim(u, w) = \cos \vec{u}, \vec{w} = \frac{\vec{u} \cdot \vec{w}}{\|\vec{u}\| \cdot \|\vec{w}\|},
\]
where the ratings of $a$ and $u$ are described as vectors of ratings $\vec{a}$ and $\vec{u}$, respectively.

The most prominent memory-based CF algorithm is called $k$ Nearest Neighbours ($k$NN) and can be divided into three phases. In the first phase a neighbourhood of the user is build. This neighbourhood consists of those users, whose ratings are most similar to the user’s ratings. Using the ratings of the neighbourhood a User-Item Matrix is generated, where an entry $(x,y)$ holds the rating of user $x$ for item $y$. In phase two the recommender algorithm aggregates the ratings of the neighbours for items the user has not rated yet. The resulting prediction is computed by averaging the ratings of the neighbourhood. In the last phase the top $N$ recommendations are displayed to the user. The main disadvantages of the memory-based algorithms are the lack of scalability and the sparsity of the generated matrix, making an accurate prediction difficult. [MKR04]

While memory-based methods have been used since the early days of recommender systems, model-based filtering has recently attracted more attention. So-called latent factor models are based on creating a model of both users and items to predict users’ rating for unrated items. The model is build by characterising users and items on multiple factors, called dimensions. A widely used approach using model-based filtering is called Matrix Factorisation (MF) and works as follows. [KBV09]

In general, ”matrix factorization models map both users and items to a joint latent factor space of dimensionality $f$, such that user-item interactions are modelled as inner products in that space” [KBV09]. Each item $i$ and each user $u$ is described by a vector $q_i \in \mathbb{R}^f$ and $p_u \in \mathbb{R}^f$, respectively. The values of $q_i$ denote the degree the item $i$ possesses the respective factor. The same holds for $p_u$ and user $u$ and thus the approximate rating $\hat{r}_{ui}$ for an unrated item $i$ is estimated via

$$\hat{r}_{ui} = \langle q_i^T p_u \rangle. \quad (2.3)$$

The system computes the factor models $p_u$ and $q_i$ by minimising the squared error function

$$\min \sum_{(u,i) \in \kappa} (r_{ui} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2), \quad (2.4)$$

where $\kappa$ is the set of $(u,i)$ pairs that the user’s rating $r_{ui}$ is known for, and $\lambda$ is a regularisation parameter. A way to minimise the squared error function is called stochastic gradient descent. This algorithm iterates over all known ratings, predicts $r_{ui}$ and computes the prediction error $e_{ui}$ using

$$e_{ui} = r_{ui} - q_i^T p_u. \quad (2.5)$$

Afterwards, the profiles $p_u$ and $q_i$ are updated by

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u) \quad (2.6)$$

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i) \quad (2.7)$$

where $\gamma$ is the learning rate and $\lambda$ the regularisation parameter.
2 Background

2.2.4 Hybrid Filtering

Often two filtering algorithms are combined to a hybrid filtering algorithm, in order to benefit from the respective advantages of both algorithms. Common combinations are CF with either demographic or content-based filtering, to help CF with the cold-start issue. The cold-start issue arises, when a new item is added or a new user joins the network. Until the new item has enough ratings or the new user has rated enough items for the recommender to provide accurate recommendations, demographic or content-based filtering help by compensating the lack of ratings.

2.2.5 Privacy Aspects

Friedman et al. [FKV+15] summarise the privacy risks, which are imposed by recommender systems. These privacy aspects are divided into direct access to data and inference from user preference data.

Direct access to data. The first risk is unsolicited data collection. Unsolicited data collection means that a service provider collects more data than necessary for providing the service in order to use them for other purpose or even sell them. This leads to the next threat, called sharing data with third parties.

Some online services outsource the recommendation process to third parties, which are specialised on generating recommendations. Although the transmitted profiles are anonymised, the control over the data is lost and de-anonymisation attacks are possible. More alarming is the fact that many companies sell personal user data to data brokers for profitable reasons. Another not negligible aspect is unsolicited access by employees, which may spy on private data of people they know or even famous personalities in order to harm them.

Inference from user preference data. Through machine learning techniques exposure of sensitive information (e.g. ethnicity, religious or political views) is possible and could harm the user. Inferring of user data may also result in targeted advertising, which became popular when Facebook introduced a targeted advertising feature, called Facebook Beacon. Beacon allowed third party websites to access private user profiles for providing personalised advertisements.

Another novel aspect is price discrimination in e-commerce based on location or characteristics of a customer (e.g. estimated salary).

Note that the mentioned risks only regard to the recommender system and risks imposed by other users or external entities are omitted because this work does not focus on that part, although some of these privacy issues are addressed as well. Furthermore we aim at technical solutions for privacy, thus leaving human aspects and perceptions of privacy out of consideration.


3 Related Work

This Chapter describes how other researchers have approached the explained privacy issues. In Section 3.1 we discuss approaches which aim to enhance privacy in OSNs. In Section 3.2 we point out approaches to enhancing privacy in recommender systems.

3.1 Distributed Social Networks

In this Section we only focus on approaches which mitigate the risks of attacks on the user by the OSN itself, as explained in Section 2.1.1. There, we focus on two approaches based on a decentralised peer-to-peer architecture in order to focus on decentralised recommender systems in Section 3.2.

Cutillo et al. [CMS09] propose a decentralised social network based on real-life trust, called Safebook. In contrast to known social networking services (SNSs), like Facebook or XING\textsuperscript{10}, Safebook is based on a peer-to-peer architecture. Each new user has to get an invitation of an user already using Safebook and then starts with creating her identity. By means of a trusted identification service the new user generates a node identifier as well as a pseudonym, based on a keyed hash function on the set of properties that uniquely identify the user in real-life. As next step, the user builds a concentric structure around his node (hereafter called core), called Matryoshka. Each ring of the Matryoshka consists of nodes interlinked according to the real-life trust between the users of these nodes. The innermost shell is called Mirror and those nodes store the encrypted data of the core. The outermost shell is called entrypoint and acts as a gateway for all demands on the core, holding a distributed hash table (DHT) with a lookup key for the user. Thus Matryoshkas enhance privacy based on "hop-by-hop" trust. For data publication the user stores the encrypted data on the Mirror nodes. Real-time messages are handled by the core, data requests can be responded by the mirrors.

Taking advantage of real-life trust and data distribution, Safebook prevents direct access and inference from user preference data by the OSN. Although the authors do not provide a evaluation of the approach, they claim that it is feasible.

Jahid et al. [JNM+12] propose a decentralised architecture for privacy in OSN, called DECENT. In the proposed architecture each post is defined as a container object, consisting of the main content and a list of comments. Each data object has three access policies: read,

\textsuperscript{10}https://www.xing.com/
append and write. The first two are attribute-based, whereas the write policy is identity-
based and all policies are defined by the object owner at the time of creation and stored as
metadata of the object. Attribute-based encryption (ABE) is a public-key encryption based
on policies described through attributes. Only users, who have the appropriate attributes
assigned, are able to decrypt the object. The storage of the data relies on distributed hash
tables as in Safebook, however, the DHT nodes are not based on real-life trust. Hence, to
ensure integrity, each object reference stores the write-policy signature key $SPK$ of the
author. Furthermore, each object is encrypted using a symmetric key, which is in turn
encrypted with ABE.

Suppose a new user, Anna, joins DECENT. First she creates an object for her profile and one
for her wall, encrypts both with a symmetric key and stores the ABE encrypted symmetric
key along with the ABE policies as well as the $SPK$ in the object reference. Then both
objects are stored with random IDs (referred to as object ID) in the DHT. A root object
contains the references to profile and wall and is stored similarly. If a user becomes her
friend, she generates him a new ABE key and sends him the key along with the attributes
and her root object. As relationships are asymmetric, the friend might give her different
attributes. A new status object is generated just like the profile and the reference is added
to the users wall object.

The evaluation shows that the user’s own wall with 20 statuses, 20 posts, and 20 comments
needs 90 seconds to construct. To view 40 feeds of friends even takes around 215 seconds
to fetch, which is well-known to be much larger than in current OSNs.

To address this issue, Nilizadeh et al. [NJM+12] propose an extension for DECENT, using
social caching for faster data retrieval. They use a gossip-based social caching algorithm,
which works as follows.

First of all each user has a presence object, stored in the DHT, which denotes the users
online status as well as his current IP and port, if the user is online. Each online user
maintains a cache with unencrypted presence and status objects from his friends as well as
a presence table that lists all social contacts along with their presence statuses. This list of
social contacts is sorted by the number of mutual friends of the user and the contact. If
a user switches from offline to online, he updates his presence information, then selects
the top contact, obtains his presence object from the DHT and decrypts it. If the contact is
online, he contacts him and obtains unencrypted presence and update objects for all mutual
social contacts. The pulled objects are in turn cached by the user (in an unencrypted form)
and the contact table is updated. This is repeated until the user knows the presence status
of every social contact. If a user leaves the network, he updates his status information
again.

This algorithm results in a decrease of time needed to build the users newsfeed. Even with
only 30% contacts online, a newsfeed is build in less than 10 seconds in contrast to over
200 seconds without social caching.
3.2 Distributed Collaborative Filtering

The related work regarding distributed collaborative filtering is divided into approaches based on a neighbourhood (*memory-based*) and approaches based on matrix factorisation (*model-based*).

### 3.2.1 Neighbourhood-based

The first approach by Miller et al. [MKR04] is called *PocketLens* and aims to both enable portability and enhance privacy of the recommender through user control over the profile. The authors introduce a decentralised collaborative filtering algorithm as well as a comparison of five different peer-to-peer (p2p) architectures. To get recommendations each user has to create a similarity model. This similarity model is basically an item-item matrix $M$, which is generated iteratively by finding new neighbours and accessing their ratings. To reduce the model size and since the model is used for computing generations only for the user, the rows of $M$ consist only of the items, $O$, the user already has rated. The columns cover the set of items rated by the neighbours. Each neighbour delivers a vector of ratings, which is used for updating the matrix and discarded afterwards. To equalise the influence of every neighbour, the vector of ratings of every neighbour $c$ is firstly normalised to $\|c\| = 1$. Each cell $(i, j)$ of $M$ determines the similarity between the item in row $i$ and the item in column $j$, and stores four values: PartialDot, PtLenU, PtLenW and Cooccur. These values are updated for all ratings received by a new neighbour for the set of items $C$, such that $C \cap O \neq \emptyset$. Each cell $(O_i, N_j)$ with $O_i \in O \cap C$ and $N_j \in C - O$ is now updated as follows, where $u_k$ is the neighbour’s rating for item $O_i$ and $w_k$ the rating for item $N_j$:

\[
\begin{align*}
\text{PartialDot}(O_i, N_j) &= \text{PartialDot}(O_i, N_j) + u_k w_k \quad (3.1) \\
\text{PtLenU}(O_i, N_j) &= \text{PtLenU}(O_i, N_j) + u_k u_k \quad (3.2) \\
\text{PtLenW}(O_i, N_j) &= \text{PtLenW}(O_i, N_j) + w_k w_k \quad (3.3) \\
\text{Cooccur}(O_i, N_j) &= \text{PartialDot}(O_i, N_j) + 1 \quad (3.4)
\end{align*}
\]

To compute the similarity between the two items the *PocketLens* algorithm uses cosine similarity. In context with the four values stored in each cell, the similarity score is now computed as follows:

\[
\begin{align*}
\text{sim}(O_i, N_j) &= k \frac{\text{PartialDot}(O_i, N_j)}{\sqrt{\text{PtLenU}(O_i, N_j)} \sqrt{\text{PtLenW}(O_i, N_j)}} \quad (3.5)
\end{align*}
\]

where $k$ is a significance weighting factor to penalise neighbours with whom the user has only a few items in common and is computed as follows:

\[
k = \begin{cases} 
1, & \text{Cooccur}(O_i, N_j) \geq 50 \\
\text{Cooccur}(O_i, N_j)/50, & \text{otherwise}.
\end{cases} \quad (3.6)
\]
After each update process, the neighbour's ratings are discarded. To generate a recommendation list, the similarity scores of each column \( N_j \) have to be summed up and the items with the highest scores are recommended. In order to focus on the distributed algorithm, the detailed explanation of the peer-to-peer architectures is now omitted, but can be found in [MKR04].

The authors evaluated their approach using their own MovieLens dataset, but mainly focused on the differences between the different p2p architectures. However, they found that the PocketLens algorithm produces at every architecture recommendations with accuracy very close to that time current recommendation algorithms with a centralised server.

Shokri et al. [SPTH09] propose an approach where a central server generates the recommendations, but the information revealed to the server is obfuscated by distributed aggregation between the users. Each user has two profiles: an online profile (stored at the central server) and an offline one (stored by the user). Note that the central server has no access to the offline profile of an user. From time to time, a user contacts users arbitrarily in a distributed manner (the authors suggest via face-to-face communication, social networks or email) and shares a subset of his offline profile for aggregation. Periodically the offline profile is synchronised with the server, but through the aggregation the server cannot distinguish between the users' ratings and the aggregated ones, and is thus preserving privacy of the user. When receiving new recommendations from the server, the user filters, whether he has already rated the items or not.

To minimise the resulting accuracy loss, two different aggregation functions are described. To determine the cardinality \( i \) of the shared subset, the functions Similarity-based Minimum Rating Frequency (SMRF) and Similarity-based Random Selection (SRS) are using the approach by Lathia et al. [LHC07], which allows to compute the similarity between two users without revealing their actual ratings to each other. SMRF now selects the \( i \) items with minimal rating frequency, whereas SRS selects \( i \) random items. While SRS results in a better accuracy, SMRF provides more privacy, because of the fact that less frequently rated items reveal more information about the user, who has rated the item.

The authors show that even with a small average of yearly contacts (approx. 6), the privacy of the system increases drastically while having a negligible loss of prediction accuracy (approx. 2%).

Berkovsky et al. [BEKR06] examined two different privacy-enhancing techniques: obfuscation of the users' data and using a distributed hierarchical neighbourhood. They found that even with obfuscating up to 90% of the users' data, the MAE (Mean Average Error) of the recommendations is not worse than the MAE of a non-personalised recommendation algorithm. The successful obfuscation methods used are the following: Neutral, where the real rating is substitute with a neutral rating (e.g. 3, when the rating ranges from 1 to 5), Random, where the real rating is substitute with a random rating and Distribution, where the real rating is substitute with a rating following the real distribution of the ratings in the dataset.

The second technique investigated reveals that at most 50% of the peers have to be queried to reach a MAE close to the MAE achieved with 100% of the peers queried (and this regardless of the density of the dataset). Thus, they suggest a hierarchical topology, where
the peers are organised into peer-groups and for each recommendation request a so-called super-peer is selected by each peer-group. This super-peer now passes the request to a subset of peers and aggregates the responses before passing them to the requester. The peers thereby use obfuscation before sending their profile to the super-peer. The requester now generates a global recommendation by aggregating the obtained local recommendations of the super-peers. This approach enhances privacy, since the super-peer cannot reveal the real ratings of the peers because of obfuscation. Furthermore, the aggregation by the super-peer prevents that the requester can distinguish certain user profiles. The subset building within each peer-group facilitates the scalability. However, comparison to centralised recommender systems is not available.

### 3.2.2 Matrix Factorisation

Vallet et al. [VFB14] propose a semi-decentralised approach using Matrix Factorisation without user data retention of the central server. At first, the central recommender collects user ratings until a cut-off time, after which the user profiles are stored locally and the ratings are send to the recommender only for generating the recommendation and updating the server model and discarded afterwards. In the first phase the recommender collects user data and maintains the latent user and item matrices \((P, Q)\), as usual for a centralised MF approach. After this phase, the recommender switches to a semi-decentralised setting by discarding the user matrix \(P\). From now on, for every new rating, the user sends his set of ratings, \(S_u\), along with the new rating \(r_{ui}\) to the server. The server now derives the user vector \(p_u\) by solving

\[
\min \sum_{r_{ui} \in S_u} (r_{ui} - p_u q_i^T)^2 + \gamma (\|p_u\|^2 + \|q_i\|^2),
\]

where \(\gamma\) is a regularisation parameter. The item matrix \(Q\) is now updated using the latent user vector \(p_u\).

The authors describe two different update variants, called new and rated. New updates \(Q\) only using the newly added rating \(r_{ui}\) by

\[
q_i \leftarrow q_i + 2\lambda [(r_{ui} - p_u q_i^T)p_u - \gamma q_i],
\]

whereas rated uses all ratings of the user. The learning rate is denoted by \(\lambda\). When the user asks for recommendations, instead of updating \(Q\), the server predicts the ratings \(\hat{r}_{ui} = p_u q_i^T\) and generates recommendations. After this process the recommender discards \(p_u\) and \(S_u\). This results in no central storage of user data and no availability of past user ratings. Although the user has to trust the service provider discarding the users data after the computation, the authors argue that disregard by the service provider would contribute to a bad reputation. However, we have to point out that the server is able to compute estimated ratings for every item, although he does not store specific ratings. Thus the claimed privacy is questionable. Compared to a centralised MF approach on the Netflix prize dataset, rated outperforms new and achieves an root mean square error (RMSE) only 1.02% higher (2.33%) than the centralised algorithm. The RMSE is a widely used measure
3 Related Work

for accuracy and determines the statistical standard deviation of the predictions.

Since personalisation in SN is very common, Isaacman et al. [IICM11] propose a fully
decentralised approach for distributed rating prediction in user generated content streams.
The authors divide the group of users $U$ into content producers $N \subseteq U$ and content
consumers $M \subseteq U$, whereby a membership of both groups is possible. Unlike in centralised
MF, every user maintains his own latent vector, instead of a centralised server. Furthermore,
each consumer maintains $|O|$ vectors, for $O$ the possible ratings (e.g. $O = \{1, 2, 3, 4, 5\}$),
and time is divided into timeslots.
Now at each timeslot $k$ a producer generates a new content item (e.g. a news-feed entry)
and shares this with his subscribers along with his profile vector. The consumer responds
with a rating $o \in O$ and his own profile vector. The provided rating by consumer $j$ for
producer $i$ at timeslot $k$ is denoted by $r_{i,j}(k) \in O$. Now $\tilde{\pi}^o_{i,j}$, $o \in O$ models the probability
that $r_{i,j}(k) = o$ for every timeslot $k$, at which $i$ delivers content to $j$. Subsequently both
update their vector using the rating, and the profile of the opponent.

The authors evaluate the approach on the Netflix prize dataset, where the movies take the
role of content producers. Before a user rates an item, the expected rating is computed. The
difference between predicted and actual rating is used for computing the $\text{RMSE}$. The authors
show that the proposed algorithm obtains a 15% better $\text{RMSE}$ than the naïve algorithm that
guesses the average rating for each movie. The $\text{RMSE}$ compared to Cinematch\[11] is roughly
5% better. The main benefit of this approach is the fact that information is only shared
between content producer and consumer and no central instance is necessary. Additionally,
their results in terms of the $\text{RMSE}$ are very promising.

3.3 Summary

In this chapter we have shown current research on privacy preserving social networks and
privacy preserving collaborative filtering. In particular, we focused on privacy through
distribution.

The research on distributed social networks shows that there are several ways to achieve
privacy in a peer-to-peer architecture. One way is to build up the network around a user
based on real-life trust, thus reducing the need of cryptography. When the network is not
based on real-life trust, a lot of cryptographic computation is necessary and thus the time
factor is a big downside. Although a lot of time can be saved for example by caching, it is
still not as fast as centralised social networks.

When it comes to decentralised CF, finding an approach with a fully decentralised setting
is difficult. Since the good results of neighbourhood based CF, some research suggests
building such a neighbourhood for each user in a peer-to-peer network as well. Other
research suggests using the peer-to-peer network only for obfuscation of the user profiles,
while using still a centralised server for computing the recommendations. Obfuscation in

\[11\]The centralised Netflix recommendation algorithm.
combination with a distributed hierarchical neighbourhood is also possible. Comparable evaluations in terms of prediction accuracy are not always provided. Although matrix factorisation is the state-of-the-art in centralised CF, research points out the difficulties of applying matrix factorisation to a peer-to-peer network. While Vallet et al. [VFB14] only suggest a semi-decentralised approach with a central server, Isaacman et al. [IICM11] is the only approach with a fully decentralised matrix factorisation algorithm.
4 Use Case and Requirements

In this Chapter we will first provide a short introduction to the Use Case and then list the most important requirements as well as a comparison of the approaches described before on these requirements.

4.1 Use Case

The core functionality of OSNs like Facebook is the filtering of the content. Instead of showing all content generated by a users’ friends or pages the user likes, an algorithm filters the content, such that only the most relevant content is displayed and thus the user spends more time on the website. This algorithm can be seen as an recommender system, where the displayed content is the recommendation. As in every recommender system, the recommendations are based on the users’ preferences, in this case collected by analysis of how the user interacts with specific content. If now, due to the above mentioned privacy issues, the social network is based on a p2p network, there is no central server to provide recommendations. Moreover, the items in user generated content streams (e.g. newsfeed entries) are, unlike for example movies, not static, but highly dynamic, which makes it difficult to provide accurate recommendations. In the next section we will present requirements for such a recommender system to work on a distributed OSN.

4.2 Requirements

Based on the explained Use Case and the privacy issues described in Section 2, we derive the following requirements for our implementation.

* Distribution. In order to enhance privacy, the implementation does not rely on a central authority and central profile storage. Instead the recommender algorithm works in a distributed fashion. Hence, each user has to maintain its own profile and compute his recommendations. The necessary information has to be gathered from the distributed network.

* Accuracy. Crucial for user experience are good recommendations. Thus the implementation should compete with centralised approaches in terms of prediction accuracy.

* Performance. As we have seen in Section 3.1 the critical factor of decentralised OSNs is
time, thus the implementation should not affect the performance, even at large scale, any further.

### 4.3 Comparison

A comparison of the approaches described in Section 3.2, regarding to the above mentioned requirements, is provided in Table 4.1. A checkmark indicates a fulfilled requirement, whereas a checkmark in parenthesis indicates an unevaluated requirement. No checkmark at all denotes an unfulfilled requirement.

The approach by Shokri et al. [SPTH09] relies on a central server and only the obfuscation process is distributed. Thus distribution is not given, although the accuracy is good. The same applies to the MF based approach proposed by Vallet et al. [VFB14]. The accuracy is good, but depends on a semi-decentralised setting with a centralised server, where privacy cannot be ensured. On the contrary, the approach by Berkovsky et al. [BEKR06] is based on a distributed setting, but lacks of prediction accuracy.

Both the requirements distribution and accuracy are fulfilled for the algorithms by Miller et al. [MKR04] and Isaacman et al. [IICM11]. The algorithm of the former operates in a p2p network in the form of an ItemKNN algorithm, whereas the other algorithm applies MF on distributed user generated content streams. The accuracy of both is acceptable, but the MF approach is more up-to-date and the results are more meaningful.

The approach by Miller et al. is the only one, where the time needed for computing the recommendations has been measured.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Distribution</th>
<th>Accuracy</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miller et al.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Shokri et al.</td>
<td></td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>Berkovsky et al.</td>
<td>✓</td>
<td></td>
<td>(✓)</td>
</tr>
<tr>
<td>Vallet et al.</td>
<td></td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>Isaacman et al.</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
</tbody>
</table>

**Table 4.1**: Comparison of the approaches with focus on the requirements.
5 Conceptual Approach

Since the approaches proposed by Miller et al. [MKR04] and Isaacman et al. [IICM11] meet all desired requirements, both are good candidates for implementation. However, because the latter one is especially designed for user generated content streams, it is optimal for the use case. As no source code of this approach is available, we will reimplement the algorithm.

In the first section of this chapter we explain the algorithm. In the second section we present the state-of-the-art algorithm for comparison.

5.1 Distributed Learning Algorithm

As shortly explained in Section 3.2.2, the approach by Isaacman et al. [IICM11] divides the group of users into content producers and content consumers. Furthermore, they divide time into timeslots and at every timeslot $k$ consumers receive content from their subscribed producer.

To formulate the rating prediction problem as distributed matrix factorisation, each content producer $i \in N$ maintains a latent vector with dimension $d$ (number of factors), referred to as $p_i \in [0,1]^d$. The consumption profile maintained by consumer $j$ consists of $|O|$ vectors, $q_{oj}^j \in \mathbb{R}^d_+$ for all $o \in O$. Thus we are able to compute a rating distribution. When producer $i$ shares content with his subscribers he sends his latent vector $p_i$ with the content. The consumer $j$ can now compute the estimated probability of rating $o \in O$, $\pi_{ij}^o$, as follows:

$$\pi_{ij}^o = \langle p_i, q_{oj}^j \rangle.$$ (5.1)

The matrix factorisation goal now results in finding vectors $p_i, q_{oj}$, which minimise the prediction error. More formally,

$$\text{minimise } E = \sum_{i \in N, j \in M} \lambda_{ij} \sum_{o \in O} |\hat{\pi}_{ij}^o - \pi_{ij}^o|^2,$$ (5.2)

where $\lambda_{ij}$ is the probability that $i$ delivers an item to $j$. To apply this optimisation problem to the distributed learning algorithm, $j$ sends the rating and his latent vector back, after rating the content item. Now both update their vectors as follows:

$$p_i \leftarrow p_i + \gamma \sum_{o \in O} (1_{r_{ij}=0} - \langle p_i, q_{oj}^j \rangle) q_{oj}^j$$ (5.3)

$$q_{oj}^j \leftarrow q_{oj}^j + \gamma (1_{r_{ij}=0} - \langle p_i, q_{oj}^j \rangle) p_i, \quad o \in O$$ (5.4)
where $\gamma = \gamma(k)$ is the learning rate. At the end of every timeslot $i$ and $j$ have to normalise their profiles such that the following conditions are satisfied:

$$\sum_{f=0}^{d} p_{i,f} = 1, \text{ with } p_{i,f} \geq 0, \text{ and}$$

$$\sum_{o \in O} q_{j,f}^o = 1, \text{ with } q_{j,f}^o \geq 0.$$  \hspace{1cm} (5.5) \hspace{1cm} (5.6)

This is done by

$$p_i \leftarrow \Pi_{D_1}(p_i), \text{ and}$$

$$q_j \leftarrow \Pi_{D_2}(q_j).$$  \hspace{1cm} (5.7) \hspace{1cm} (5.8)

where $\Pi_{D_1} : \mathbb{R}^{|F|} \rightarrow D_1$, $\Pi_{D_2} : \mathbb{R}^{|O| \times |F|} \rightarrow D_2$ are the orthogonal projections to $D_1$ and $D_2$, respectively. $D_1$ and $D_2$ are the sets of profiles that satisfy (5.5) and (5.6), respectively. The algorithm we use for the projection is described in Section 5.1.2.

**Example.** Suppose we have two users in an OSN, Anna and Ben, and consumer Ben is a subscriber to producer Anna’s content. Ben maintains his two consumer vectors $q_{ben}^1 = [0.2, 0.1, 0.4]$ and $q_{ben}^2 = [0.8, 0.9, 0.6]$ for the ratings $o \in \{1, 2\}$, where 1 is a negative and 2 a positive rating. Anna maintains her producer vector $p_{anna} = [0.2, 0.5, 0.3]$. Now Anna shares a new picture, which means she sends this picture to all her subscribers, including Ben, along with her producer vector $p_{anna}$. When Ben receives the picture and Anna’s vector, he can compute his estimated rating for the picture as follows:

$$\text{PredictedRating}(i,j) = \sum_{o \in O} o \cdot \langle q_{j,f}^o, p_{i} \rangle,$$  \hspace{1cm} (5.9)

for producer $i$ and consumer $j$. In particular, $\text{PredictedRating}(Anna, Ben) = \sum_{o \in O} o \cdot \langle q_{ben}^o, p_{anna} \rangle = 1 \cdot 0.21 + 2 \cdot 0.79 = 1.79$. As the predicted rating is close to rating 2, the picture will be displayed to Ben. After looking at the picture, Ben sends his rating $r_{ben,anna} = 2$ and his consumer vectors $p_{ben}$ to Anna. Now both can update their vectors according to (5.3) and (5.4) and normalise the vectors afterwards by (5.7) and (5.8).

$\text{Figure 5.1:}$ Example for the Distributed Rating Prediction algorithm in a OSN with two users.

The pseudo code of the distributed learning algorithm by Isaacman et al. [IICM11] is shown in Algorithm 5.1. Because of the importance, the learning rate is described in the next section.
5.1 Distributed Learning Algorithm

Algorithm 5.1 Distributed learning algorithm from [IICM11]

1: Producer $i$ at timeslot $k$:
2: $i$ generates new content item
3: $\gamma \leftarrow \gamma(k)$
4: for every pair $i,j$ s.t. $a_{i,j}(k) = 1$ do
5: $i$ sends its item and $p_i$ to $j$.
6: $i$ receives $r_{i,j}$ and $q_j$ from $j$.
7: $p_i \leftarrow p_i + \gamma \sum_{o \in O} (\mathbb{1}_{r_{i,j}=0} - \langle p_i, q^o_j \rangle)q^o_j.$ // update
8: end for
9: $p_i \leftarrow \prod_{D_1} (p_i)$ // normalisation

10: Consumer $j$ at timeslot $k$:
11: $\gamma \leftarrow \gamma(k)$
12: for every pair $i,j$ s.t. $a_{i,j}(k) = 1$ do
13: $j$ receives item and $p_i$ from $i$.
14: $j$ rates item with $r_{i,j} \in O$.
15: $j$ sends $r_{i,j}$ and $q_j$ to $i$.
16: for every $o \in O$ do
17: $q^o_j \leftarrow q^o_j + \gamma(\mathbb{1}_{r_{i,j}=0} - \langle p_i, q^o_j \rangle)p_i.$ // update
18: end for
19: end for
20: $q_j \leftarrow \prod_{D_2} (q_j)$ // normalisation

5.1.1 Learning rate

We found that the learning rate is crucial for the resulting $\text{RMSE}$. If the learning rate is too large, unexpected deviation of ratings may significantly affect the factors. Conversely, if the learning rate is too low, the algorithm needs too much time for convergence and new trends are adapted too slowly.

In their paper, Isaacman et al. only assume that the learning rate $\gamma(k)$ fulfills the conditions $\gamma(k) \geq 0$, $\sum_{k=1}^\infty \gamma(k) = \infty$, and $\sum_{k=1}^\infty (\gamma(k))^2 < \infty$. We see that the learning rate depends on the current timeslot $k$, whose length is not specified in the paper as well. With no information about the length of a timeslot and the learning rate we achieved the best results with a constant learning rate.

5.1.2 Normalisation

After each update process, the latent item factors and the latent user factors have to be normalised in order to fulfill the requirements (5.5) and (5.6). For these projections we used the finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$, proposed by Michelot [Mic86], which works as described below.
Preliminaries. The projection we want to compute can be written as
\[
\inf_{x} \frac{1}{2} \|x - c\|^{2}_{n}, x \in K,
\]
with \( K = \{x \in K | \sum_{i=1}^{n} x_i = 1, x \geq 0\} \). Furthermore we need the following definitions:
\( I_n = \{1, 2, ..., n\} \),
\( V = \{x \in X | \sum_{i=1}^{n} x_i = 1\} \).

For an arbitrary subset \( I \) of \( I_n \), let
\( X_I = \{x \in X | x_i = 0, \text{ for all } i \in I\} \)
\( V_I = X_I \cap V \),
\( K_I = X_I \cap K \),
\( n_I = \dim(X_I) \),
where \( X_I \) is a linear subspace of \( X \).

Algorithm 5.2 Projection of a point onto the canonical simplex of \( \mathbb{R}^{n} \) [Mic86]

```
procedure PROJECT(c)
    I = ∅, x = c // initialisation
    for i < dim do
        ˜x = PV_i(x)
        if ˜x ≥ 0 then
            return ˜x
        else
            I ← I ∪ {i | ˜x_i < 0}
            x = PX_I(˜x)
        end if
    end for
end procedure

procedure PV_i(x) // projection of x
    for all i do
        ˜x_i ← \[
            \begin{cases} 
                x_i - \frac{\sum_{i \in I} x_i - 1}{n_I}, & i \notin I \\
                0, & i \in I 
            \end{cases}
        \]
    end for
end procedure

procedure PX_I(˜x) // projection of ˜x
    for all i do
        x_i ← \[
            \begin{cases} 
                ˜x_i, & ˜x_i \geq 0 \\
                0, & otherwise 
            \end{cases}
        \]
    end for
end procedure
```

Algorithm. The input for the algorithm is the vector \( c \), which has to be normalised. The set \( I \) is initialised as the empty set and the vector \( x \) as \( c \). For each iteration we compute
the projection $P_{V_1}(x)$. Is the resulting vector $\tilde{x}$ positive ($\tilde{x} \geq 0$), which means $x_i \geq 0$ for all $i \in \{1, ..., n\}$, the computation is finished and $\tilde{x}$ is the desired projection of $c$. Otherwise we extend $I$ by $\{i|\tilde{x}_i < 0\}$ and replace $x$ by the projection $P_{X_I}(\tilde{x})$. The algorithm ensures fast computation, since it converges with at most $n$ iterations.

Algorithm 5.2 displays the pseudo-code for the algorithm.

5.2 Comparison Algorithm

Besides the baseline algorithm ItemAverage, which always predicts the average rating for every item, we compare our results to a state-of-the-art algorithm, called BiasedMF, as well.

5.2.1 BiasedMF

The BiasedMF algorithm proposed by Koren et al. [KBV09] is a centralised matrix factorisation approach. It enhances the standard matrix factorisation by taking a user and item bias into account. The user bias $b_u$ indicates the general deviation of the users ratings from the overall average rating $\mu$ and the item bias $b_i$ indicates the deviation of an item from the overall average rating. Remember that the standard MF algorithm computes the estimated rating $\hat{r}_{ui}$ for user $u$ and item $i$ by $\hat{r}_{ui} = \langle q_i^T p_u \rangle$ (2.3). When we use the biases for better accuracy, the estimated rating is computed as follows:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$$

(5.11)

The squared error function (2.4) has to be adapted as well and is now:

$$\min \sum_{(u,i) \in \kappa} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2 + b_i^2 + b_u^2)$$

(5.12)

The parameters of $b_i$ and $b_u$, as well as the latent factors, are learned by the gradient descent algorithm, as described in 2.2.3.
6 Implementation

This chapter describes how we implemented the algorithms we presented in the preceding chapter. In the first section we describe the general infrastructure, whereas we explain the organisation of our code in the second section.

6.1 General Infrastructure

We used the programming language Java. Furthermore, we used the open source Java library LibRec 2.0\textsuperscript{12}, which provides more than 70 implementations of recommender algorithms and allows for efficient implementation of own recommender algorithms. In particular, the provided implementations include several Matrix Factorisation based approaches. Moreover LibRec supports a few commonly used datasets for evaluating recommender systems like the MovieLens dataset\textsuperscript{13}.

LibRec provides an abstract class \texttt{MatrixFactorizationRecommender}, which extends the abstract class \texttt{AbstractRecommender} from which all recommender classes inherit. \texttt{MatrixFactorizationRecommender} provides variables like the learning rate as well as the number of factors and of course the matrices \texttt{userFactors} and \texttt{itemFactors}. Because our approach not only needs one vector for each user, but \(|o|\) vectors for each user (one vector for every possible rating), we used an array of user factor matrices instead of one matrix.

Furthermore, LibRec supports reading datasets from (csv) files and convert them into a \texttt{trainMatrix} and a \texttt{testMatrix}. This is done with the class \texttt{TextDataModel}. However, the Netflix prize dataset we used for our evaluation, which is presented in the following chapter, is stored in a different format, which the class \texttt{TextDataModel} is not able to handle. So we added a class \texttt{NetflixAdapter}, which converts the Netflix dataset as well as the probe dataset, which we used for our second experiment, into a format, which can be read by \texttt{TextDataModel}. Moreover, for our first experiment we do not need a \texttt{testMatrix}, so we configured LibRec to put all ratings into the \texttt{trainMatrix}. In our class \texttt{DRPMFRecommender} we then sorted the ratings of the \texttt{trainMatrix} chronologically.

We used Eclipse neon.3 (4.6.3) as our development environment.

\textsuperscript{12}Librec: https://www.librec.net/
\textsuperscript{13}MovieLens: https://movielens.org/
6.2 Code Organisation

We implemented 7 classes in three packages. We explain briefly each package and what each class does. The three packages are called **recommender**, **experiment** and **adapter**.

The **recommender** package contains three classes for different recommender algorithms. The class **DRPRecommender** provides the implementation of the algorithm by Isaacman et al. [IICM11], which is described in the preceding chapter. The class **ChronBiasedMFRecommender** is a modification of the LibRec implementation of the **BiasedMF** algorithm by [KBV09]. It is modified in order to fit the decentralised experiment design, which is described in the following chapter. The third class **ChronItemAverage** is a implementation of the **ItemAverage** algorithm, again in order to fit the first experiment design.

The **experiment** package contains two classes for the experiments described in the following chapter. The class **DecentralisedDesign** contains the code for executing the first experiment design. The class **CentralisedDesign** contains the code for executing the second experiment design.

The **adapter** package includes two classes. The class **NetflixAdapter** serves as converter of the Netflix Prize dataset and the probe dataset. After converting the datasets, LibRec is able to handle them. The **Converter** class implements the main method and is called to execute the **NetflixAdapter**.
7 Experiment

This Chapter describes the evaluation process of the implemented approach. The first section describes the experiment structure, the dataset and the metrics used to evaluate the performance of the algorithm. In the second section we present the results of the experiment. The third section provides a discussion of the results.

7.1 Experiment Structure

7.1.1 Dataset

The used dataset was released by Netflix in 2006 for the challenge of finding the best collaborative filtering algorithm. The ratings were collected of the Netflix users between October 1998 and December 2005. The dataset contains 100,480,507 ratings of 480,189 users for 17,770 movies. The ratings range from 1 (star) to 5 (stars) and are stored as quadruples along with the user ID, movie ID and the date of grade. Note that the dataset is modified such that the identity of the users is not traceable. The Netflix baseline algorithm Cinematch achieves an RMSE of 0.9525 on the test set\textsuperscript{14}. The winning team of the Netflix prize has an RMSE of 0.8567\textsuperscript{15}.

7.1.2 Metrics

The most popular metric to measure the predictive accuracy of a recommender system is called root mean square error (RMSE).

\textbf{RMSE} is a metric to measure the difference between the predicted rating and the users true rating and it is computed as follows:

\[
RMSE = \sqrt{\frac{\sum_{i=1}^{n} (\hat{r}_i - r_i)^2}{n}},
\]

(7.1)

where \(\hat{r}\) is the predicted and \(r\) the actual rating, \(n\) denotes the number of ratings. Furthermore we evaluate the \textbf{time} needed for computing the recommendations.

\textsuperscript{14}http://www.netflixprize.com/prize_4.html
\textsuperscript{15}http://www.netflixprize.com/leaderboard.html
7.1.3 Experiment Design

In the next section we describe the method, Isaacman et al. [IICM11] used to evaluate their approach. Then we present the setting for an evaluation, which is normally done to evaluate recommender algorithms.

Isaacman Design

First of all, we sort all ratings chronologically in order to simulate a realistic environment, where the profiles are adapted as user rate movies. The vectors of users and items are initialised randomly with small Gaussian values. Now prior to each rating we compute the estimated rating for user $j$ and item $i$. For the distributed rating prediction (DRP) algorithm by Isaacman et al. the predicted rating is computed according to (5.9). The predictions for the BiasedMF algorithm are computed following (5.11). Afterwards, we update the user factors as well as the item factors and use the difference between the predicted and the real rating for the RMSE computation, as defined in (7.1). The update process for the DRP algorithm is described in (5.3) and (5.4), while the profiles of the BiasedMF algorithm are updated according to (2.6) and (2.7).

To neglect the training period required by the algorithms for representative results, early predictions are not included in the RMSE computation. In particular, we measure two types of training periods. For the first, we only use predictions, where either the user OR the item has been updated at least $k$ times. Secondly, we only use predictions, where both the user AND the item have been updated at least $k$ times. We refer to them as the OR and the AND training period, respectively.

In order to compare the DRP against the BiasedMF algorithm in the same manner, we changed the LibRec implementation of the BiasedMF algorithm in such a way that the evaluation works similar as for the DRP algorithm. We have to point out that the BiasedMF algorithm depends on global information and only works this way, because we evaluate in a centralised setting. It can not be used this way for a decentralised network.

Standard Design

The standard evaluation method for matrix factorisation recommender systems differs from the above explained. Normally, the Netflix dataset is used to train the recommender system using the gradient descent algorithm for several iterations.

Afterwards, the trained model is evaluated on the probe dataset. The probe dataset has the same statistical properties as the qualifying dataset and contains 1,408,395 ratings. The decisive difference between the qualifying dataset, which is used by the jury to evaluate the performance of the recommender systems, is that the ratings are already contained in the normal dataset. So evaluating on the probe dataset allows for good comparison between different recommender algorithms. Because Isaacman et al. [IICM11] did not evaluate their algorithm on the probe dataset, a comparison to different algorithms is not possible.
Hence, we evaluate the Isaacman approach on the probe dataset in order to compare the result with the BiasedMF algorithm. Same as for the Isaacman design, we initialise the vectors of users and items, and the user and item matrix, respectively, randomly with small Gaussian values. Now, we iterate several times (e.g. 20-30) over the dataset, and, same as described for the other design, we predict the rating and update the vectors and matrices, respectively. When the model is trained, we predict the rating for each entry of the probe dataset according to (5.9) and (5.11) for DRP and BiasedMF, respectively. Finally, the RMSE is measured using the difference between the predicted and the actual rating, as defined in (7.1).

**Differences between the Designs**

The Isaacman and the standard design differ in the way the latent factor model is trained and in the way the trained recommender system is evaluated. For the Isaacman design we take the whole Netflix dataset (100,480,507 ratings) and train the model by chronologically iterating over the ratings for a single time, whereas for the standard design we iterate over the Netflix dataset without the probe dataset (99,072,112 ratings) for several iterations. While, at the Isaacman design, we compute the RMSE of the algorithms during the training of the model, we compute the RMSE on the probe dataset (1,408,395 ratings), after training the model, at the standard design.

**7.2 Experiment Evaluation**

We executed the experiment on a server with an Intel Westmere-EX X5675 processor with 3.06GHz (6 cores) and 96GB RAM.

**7.2.1 Isaacman Evaluation**

Figure 7.1 depicts the effect of different training periods on the RMSE. The parameters for the BiasedMF approach are the following. The regularisation parameter of users and items is 0.02, the learning rate is 0.04. The learning rate for the DRP algorithm ranges from 0.04 to 0.08, and the the dimension for both algorithms is 10. As expected, the RMSE of the ItemAverage algorithm does not change for different training periods and remains at 1.01. Similarly the RMSE of the DRP algorithm with the OR training period does not significantly change with a higher training period, instead it stays about 0.94. The DRP approach with the AND training period is the only one, which shows a significant decrease of RMSE for a higher training period, down to an RMSE of roughly 0.85 for \( k = 300 \). We explain this by the fact, that the average movie is rated more than 5000 times, whereas the average user only rated over 200 movies. Thus the higher training
Experiment

The results of the BiasedMF algorithm with a similar execution like the DRP algorithm, therefore referred to as chronBiasedMF, backs up our assumption. The OR evaluation, as explained by Isaacman et al., has no influence on the RMSE ($\approx 0.87$). However, the AND evaluation leads to a decrease of the RMSE, down to approx. 0.81. Thus, we will talk only of the AND training period hereafter.

Although we see that the centralised algorithm performs clearly better, the results of the decentralised are acceptable. With a training period of only 50 views, the results are vastly better than those we get with the ItemAverage algorithm. Considering the lack of

![Figure 7.1: RMSE vs. training period](image_url)
information caused by the decentralised setting, we can conclude that the predictions are nevertheless quite accurate.

In Figure 7.2 we see the effect of different dimensions on the RMSE. The RMSE of the chronBiasedMF algorithm starts at approx. 0.89 only using one factor and achieves roughly 0.87 for only 10 factors. However, more than 10 dimensions do not lead to a further decrease of the RMSE. On the other hand the RMSE of the DRP algorithm (with training period 0) starts at approx. 0.99 only using one factor, achieves the best result of approx. 0.93 for 3 factors and remains at approx. 0.94 for dimensions greater than 6. This indicates that a small number of factors is required to recognise all aspects of the dataset, while too much factors, in contrast to the chronBiasedMF algorithm, do not contribute to better predictions.

Moreover, the DRP algorithm with a training period of only 100 views is as good as the chronBiasedMF without training period, whereas a training period of 200 and 300 views outperforms the chronBiasedMF.
### 7 Experiment

<table>
<thead>
<tr>
<th>Approach</th>
<th>Time</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRP 4 factors</td>
<td>2 min 34 s</td>
<td>0.8813</td>
</tr>
<tr>
<td>DRP 10 factors</td>
<td>4 min 35 s</td>
<td>0.8883</td>
</tr>
<tr>
<td>DRP 20 factors</td>
<td>6 min 57 s</td>
<td>0.8906</td>
</tr>
<tr>
<td>DRP 30 factors</td>
<td>11 min 5 s</td>
<td>0.8937</td>
</tr>
<tr>
<td>chronBiasedMF 10 factors</td>
<td>58 s</td>
<td>0.8410</td>
</tr>
<tr>
<td>chronBiasedMF 20 factors</td>
<td>1 min</td>
<td>0.8387</td>
</tr>
<tr>
<td>chronBiasedMF 30 factors</td>
<td>1 min 2 s</td>
<td>0.8383</td>
</tr>
<tr>
<td>ItemAverage</td>
<td>31 s</td>
<td>1.012</td>
</tr>
</tbody>
</table>

**Table 7.1**: Evaluation of time (training period = 50).

When we measure the time the algorithms need to iterate over the whole dataset and train the model, we see that the DRP algorithm needs significantly more time than the chronBiasedMF algorithm. Since the ItemAverage algorithm does not need much computation, the rapidity was expected. While the difference between 10 and 30 factors is only four seconds for the chronBiasedMF algorithm, the DRP algorithm needs more than twice as much time with 30 factors as with 10 factors. Furthermore, the DRP algorithm with 10 factors requires more than four times more time than the chronBiasedMF algorithm with 10 factors. Even with only four factors the DRP algorithm takes twice as much time compared to the chronBiasedMF algorithm. Hence, more than four factors do not lead to better results and need significantly more time.

However, this time consumption is no issue, since the algorithm is designed to work in a decentralised setting, where the user and item profiles are trained when user rate movies, which is a period of years. Furthermore the computation process is distributed among all users, which makes the time negligible.

#### 7.2.2 Standard Evaluation

As described above, we evaluated the algorithms in the standard way for matrix factorisation algorithms as well. To achieve this, we trained both the algorithms employing the gradient descent algorithm for 25 iterations. The parameters for the BiasedMF approach are the following. The regulation of users and items is 0.02 and the learning rate is 0.005. The learning rate for the DRP algorithm is 0.02. The results are displayed in Figure 7.3. We know that the ItemAverage algorithm achieves an RMSE of 1.05 on the test dataset, whereas the Netflix algorithm Cinematch reaches an RMSE of 0.95.

Compared to these results, we see that the DRP performs clearly better than the ItemAverage, but cannot surpass the Cinematch algorithm. However, the BiasedMF algorithm outperforms the Netflix algorithm.
Nevertheless, note that the DRP algorithm is not designed to work this centralised way - its strength lies in predicting accurate ratings in a peer-to-peer network, where only two parties interact at a time.

7.3 Discussion

In Figure 7.1 we have seen that a training period of 50 views improves the RMSE by nearly 7 percent in the case of the DRP algorithm, in contrast only by 3.5 percent in the case of the BiasedMF algorithm. This indicates that the BiasedMF algorithm overcomes cold-start situations better than the DRP algorithm, since the RMSE of the BiasedMF decreases nearly linear. This shows that later updates are as important as early ones, and thus the BiasedMF algorithm performs good from scratch.

When comparing the results of the Isaacman and the standard evaluation, we see that, even with a training period of 0, the RMSE of the Isaacman design is better than the best RMSE of
the standard evaluation (0.94 vs. 0.97). One possible explanation is that the probe dataset contains many cold-start situations, in order to test the robustness of the recommenders. As we have seen in the Isaacman evaluation, the DRP algorithm can be improved concerning the cold-start problem.

As Bell et al. [BK07] have pointed out, the best pure centralised MF algorithm achieves an RMSE of 0.908 on the Netflix testset (similar to probe dataset) with the standard design. Hence, for better accuracy, centralised MF approaches nowadays include a combination of several filtering approaches, which will be necessary for the DRP algorithm as well. [Kor08] and [Kor10] show how to combine centralised MF with a neighbourhood-model or temporal dynamics.

### 7.4 Summary

In this chapter, we presented our experiment. We evaluated the DRP algorithm in two different ways in order to make a fair comparison to a centralised state-of-the-art algorithm possible.

The first evaluation process was similar to the evaluation presented by Isaacman et al. [IICM11] This evaluation shows the superiority of the DRP algorithm to the simple algorithm, which always predicts the average rating for every item. However, the centralised BiasedMF algorithm outperforms the DRP algorithm. Furthermore we evaluated two different training periods. Comparing the results to the results presented by Isaacman et al. [IICM11], we discovered that their results do not match their description of the evaluation.

We used the second evaluation method to compare the algorithm to centralised approaches, although the DRP algorithm is not designed for this setting. The results show an improvement over the ItemAverage approach, but the DRP stands no chance against the centralised algorithms Cinematch and BiasedMF.
8 Conclusion

In this thesis we have reimplemented the matrix factorisation based distributed rating prediction algorithm by Isaacman et al. [IICM11]. The benefit of this algorithm is the ability to work in a peer-to-peer network (e.g. a decentralised social network) and personalise the user only sharing profiles between users in the network without the need of a central storage. Thus we have met the distribution requirement.

Furthermore, the algorithm fulfills the accuracy requirement, as we have shown the algorithm performs only 5% worse than the centralised MF algorithm in terms of the standard deviation of the predictions (RMSE). Moreover, we have demonstrated that the time the algorithm takes to predict over 100 million ratings is acceptable, since the workload in a decentralised setting is distributed as well.

In addition, we mitigate the privacy aspects direct access to data and inference from user preference data, which are described in Section 2.2.5. The algorithm ensures that no party has global knowledge, unless the party has interacted with the entirety of users. Hence we fulfilled our thesis goal. One drawback remaining is the exposure of preference data to other users, who may not discard the preferences after the update process.

During the evaluation we have discovered the difficulties of comparing the RMSE of the decentralised algorithm to centralised state-of-the-art algorithms.

On the one hand we used the methodology of Isaacman et al. [IICM11], which leads to great results using a training period, where we only use those predictions for measuring the error, where both the user and the item profile have been updated more than 300 times. Even with a more moderate training period of 100 views the error is only roughly 5 percent larger than the centralised BiasedMF algorithm. However, we have to state that using the evaluation, how it is described in the paper of Isaacman et al.[IICM11], we achieved different results as the ones they published. Although the paper lacks of some information about the values of the learning rate parameter and the timeslot size, we figured out some optimal values.

On the other hand, we show that the results of the Isaacman et al. [IICM11] evaluation cannot be compared to the published results of state-of-the-art algorithms. When we adapt the algorithm to make the standard evaluation possible - although the algorithm is not designed for this situation - the results are worse than the results for the Isaacman evaluation without training period. We have traced back this circumstance to the DRP algorithm having problems with the cold-start situations.

Focusing on the first evaluation methodology, we conclude that decentralised personalisation is possible. Moreover, it is feasible and not vastly worse than centralised personalisation.
Further work. For further privacy research, the encryption or obfuscation of the user profiles is certainly of major interest. The only downside of the evaluated algorithm (in terms of privacy) is the need to expose the personal profile to other users. Although a user only uses other profiles for updating his own profile and discards them afterwards, an exploit of the data cannot be prevented. During our research we have only encountered approaches that address obfuscation, like [SPTH09] or [BEKR06].

In order to further improve the prediction accuracy, research has to overcome the cold-start problem for decentralised settings. A common approach to address the cold-start problem is the use of a hybrid filtering method as described in [Kor08] and [Kor10] and could be used for the DRP algorithm as well.
Bibliography


All links were last followed on September, 2017.