Movie recommendations using matrix factorization

JAKOB IVARSSON

MATHIAS LINDGREN
Movie recommendations using matrix factorization

Jakob Ivarsson and Mathias Lindgren

Degree Project in Computer Science DD143X
Supervised by
Arvind Kumar
Examiner
Örjan Ekeberg

11 maj 2016


Abstract

A recommender system is a tool for recommending personalized content for users based on previous behaviour. This thesis examines the impact of considering item and user bias in matrix factorization for implementing recommender systems. Previous work have shown that user bias have an impact on the predicting power of a recommender system. In this study two different implementations of matrix factorization using stochastic gradient descent are applied to the MovieLens 10M dataset to extract latent features, one of which takes movie and user bias into consideration. The algorithms performed similarly when looking at the prediction capabilities. When examining the features extracted from the two algorithms there was a strong correlation between extracted features and movie genres. We show that each feature form a distinct category of movies where each movie is represented as a combination of the categories. We also show how features can be used to recommend similar movies.
Sammanfattning

# Contents

1 Introduction 1
   1.1 Purpose of this study ........................................... 1
   1.2 Problem statement ............................................. 2

2 Background 3
   2.1 Recommender System ............................................ 3
   2.2 Content-based filtering ......................................... 3
   2.3 Collaborative filtering ......................................... 4
      2.3.1 Cold start .................................................. 4
      2.3.2 Overfitting .................................................. 4
      2.3.3 Bias ......................................................... 4
   2.4 Matrix factorization ............................................ 5
      2.4.1 Stochastic gradient descent ............................... 6
      2.4.2 Alternating least squares ................................. 6
      2.4.3 Matrix factorization with bias ............................ 7
   2.5 Relevance and previous research ................................ 7

3 Method 8
   3.1 Factorization ..................................................... 8
   3.2 Parameters ....................................................... 8
   3.3 Bias ............................................................. 8
   3.4 MovieLens dataset ............................................... 8
   3.5 Evaluation ....................................................... 9
      3.5.1 Root Mean Squared Error ................................ 9
      3.5.2 Testing the algorithms .................................... 9
   3.6 Feature extraction ............................................... 9
   3.7 Finding similar movies ......................................... 10
   3.8 Convergence test ................................................ 10
   3.9 Implementation ................................................ 10

4 Results 11
   4.1 Extracted features ............................................... 11
   4.2 Recommending similar movies .................................. 12
   4.3 Comparison of algorithms ..................................... 13

5 Discussion 16
   5.1 Limitations ...................................................... 17
   5.2 Further research ................................................ 17
   5.3 Conclusions .................................................... 17
1 Introduction

Predicting what content a user wants today is important for a lot of sites on the internet in order to be competitive. Sites such as Netflix, Amazon and Youtube have developed highly sophisticated systems for recommending new and relevant content for users. These systems are one of the most valued assets of these companies as demonstrated by the Netflix sponsored competition with a prize of one million dollars to improve on their system [1]. These types of systems are collectively known as recommender systems [2].

Recommender systems can take multiple different approaches to achieve comparable results. An important task of a recommender system is to make predictions of how a user might like an item based on the user’s and other users’ past behaviour. The two most common techniques deployed are content-based and collaborative filtering. These two both have their pros and cons. Content-based filtering compare item attributes and make recommendations by finding items that are similar to what a user previously liked. Collaborative filtering on the other hand tries to find users that have similar taste and predict based on how these users interacted with different items.

Collaborative filtering uses user-item relations to make predictions for a user. This is a powerful approach as the system does not need to know anything about the items inherent attributes. Hence it can easily be applied to any dataset with user-item relations. Collaborative filtering methods can be divided into two categories; memory-based and model-based. Memory-based approaches is usually simple to implement and can provide very good prediction scores. However, memory-based techniques has been proven to be inefficient and difficult to scale with large datasets that is common in many real-world applications. In this study we will therefore investigate the model-based approach in construction of recommender systems.

Model-based collaborative filtering uses machine learning algorithms to create a model based on training data and then use the model to make predictions. This thesis will compare two implementations of matrix factorization using stochastic gradient descent. When creating the model from training data, computer inferred latent features are extracted that fits the data.

1.1 Purpose of this study

Matrix factorization techniques has been proven to be effective for implementing recommender systems, most notably in the Netflix prize competition [1]. However, most studies has been focused on making accurate predictions and not on what the models represent when fitted. Extracted latent features from the models created by matrix factorization are at first sight hard to understand. This report seeks to understand what the latent features represents by examining what the models have learned.
This study will also compare a matrix factorization implementation that uses a pure decomposition model with one that incorporates bias to create the model to see if better predictions can be achieved.

1.2 Problem statement

The goal of this report is to give a deeper understanding of matrix factorization models for recommender systems by answering the following questions:

- Can incorporation of bias into the matrix factorization model improve movie rating predictions?
- How can the latent features extracted from movie ratings by matrix factorization be interpreted?
2 Background

2.1 Recommender System

A recommender system is a system that recommends items for a user. Recommend can mean different things in different contexts. In the general case given an item in the system and/or a user of the system it is either 1. find similar items/users 2. give a prediction of how much the user will enjoy the particular item. The general recommender systems’ data include users, items that the users can interact with, item features, and user-item interactions. Example of items can be movies, songs, recipes, and products in an online shop such as clothes or electronics. The user-item interactions can consist of either unary data such as a view or purchase, binary data such as like or dislike, a numerical value such as a 1-5 rating or time viewed. Recommender systems are particularly useful in cases where there is an abundance of content that is not relevant to the user. The goal of a recommender system is often to increase the number of items sold or time spent interacting with the system, sell more diverse items and increase user satisfaction. A number of different approaches for predicting user preference and item similarity has been developed and used. They can be divided into three categories; content-based filtering (section 2.2), collaborative filtering (section 2.3) or a hybrid approach combining both content and collaborative filtering.

2.2 Content-based filtering

Content-based filtering (CB) is a method for recommender systems where the primary source of information is content known about the items in the system. This can be tags, inherent attributes such as size, length, color, name and so on. In the context of movies, content could be meta information about the movie such as director, actors, genre and release date. CB recommender systems typically make recommendations by finding items that are similar to ones the user has liked in the past. Predictions are typically made by building a profile for the user preferences based on the content of the items that the user previously liked, rated, or interacted with. New items are compared to the user’s profile and given a relevancy score for the user. While having an accurate user profile can be very effective, this approach has some limitations if the items content does not have sufficient amount of features. Because these systems will only recommend items that are similar to the ones already rated, a user can miss out on items that the user may like even though they are not very similar to what they have interacted with before. Items that may be wildly different may have some sort of usefulness together that is not directly linked to their features. Cold start is a problem but not as big as with Collaborative filtering as the system can make predictions agnostic to how many users have used the system.
2.3 Collaborative filtering

Collaborative filtering (CF) recommender systems make predictions by looking at user-item relations, without the need to have any additional information about the users or items. Two common approaches when implementing CF recommender systems are neighborhood-based CF and latent factor models [5].

Neighborhood-based methods make predictions by calculating similarities between users or items based on the user-item relations. The user oriented approach look at what other users with similar interactions when making predictions [3]. The item oriented approach, on the other hand, calculates an item to item similarity and make predictions by weighting the similar items that the user has rated before [6].

Latent factor models for movie recommendations utilize the user-item rating matrix to try to characterize both the users and items by a number of latent features. The number of factors can typically be 10-100 for each user and item. The factor vectors for a user and item can be multiplied together to give a predicted rating for the item. These factors can be seen as computer inferred features of an item or user. In the context of movies, a feature can be a genre, target age group, maybe a less obvious thing such as amount of character development or even completely uninterpretable. An effective latent factor method for recommender systems is called matrix factorization [7] which we will focus on in this report.

2.3.1 Cold start

Cold start is an even bigger problem when using a CF based approach in implementing a recommender system compared to a CB approach. As the information used is primarily the behaviour of the users of the system, a big number of behavioural interactions must take place before the system becomes useful. In this study we will focus on the interpretation of the feature model extracted from a static training data and not the usefulness in a real time system. [7]

2.3.2 Overfitting

When training a system on a set of data, it is not necessarily the case that the best fit for the actual data is the fit that is best for the training data, this is called overfitting. To combat overfitting one can during training look at what predictions the system makes for data outside of the training set and see when the system makes the best predictions.

2.3.3 Bias

As the system makes predictions solely based upon the interactions of its users, the users can introduce bias into the system. An example of behavioural specific bias would be a user that only rates 5 and 4 for movies the user like and dislike. This means that the system need to look at what a users own rating scale means to get comparable results to other
users. The effect of bias can be mitigated by normalizing the users ratings so that all the users more or less, at least seem to, behave in the same way towards the system.

2.4 Matrix factorization

Matrix factorization (MF) is a technique which computes a latent factor model of a system based on user-item interaction. These user-item interactions can be represented as a matrix with users on one axis and items on the other. In most movie recommender systems interactions are ratings by users for movies, but can be different data as well, such as implicit feedback, temporal effects, and/or confidence levels. This rating matrix is typically very sparse in real world applications as users usually only rates a fraction of all the movies in the system. MF has been showed to be able to make very good predictions even on very sparse matrices.

The matrix factorization approach reduces the dimensions of the rating matrix $r$ by factorizing it into a product of two latent factor matrices, $p$ for the users and $q$ for movies [7].

$$
\begin{pmatrix}
  r_{11} & \cdots & r_{1i} \\
  \vdots & \ddots & \vdots \\
  r_{ui} & \cdots & r_{ui}
\end{pmatrix}
\times
\begin{pmatrix}
p_1 \\
p_2 \\
\vdots \\
p_u
\end{pmatrix}
= 
\begin{pmatrix}
q_1 & q_2 & \cdots & q_i \\
\vdots & \vdots & \ddots & \vdots \\
p_i
\end{pmatrix}
\times
\begin{pmatrix}
q_1 \\
q_2 \\
\vdots \\
q_i
\end{pmatrix}
$$

$$
\mathbf{r} = \mathbf{pq}^T \quad (1)
$$

$f$ is the number of features extracted, $u$ the number of users and $i$ number of items (in our case, movies).

Each row $p_u$ is a vector of features for a user $u$ and each row $q_i$ is a vector of features for an item $i$. The product of these vectors creates an estimate of the original rating.

$$
r_{ui} = p_u q_i^T \quad (2)
$$

There are multiple ways of factorizing a matrix into multiple components, used in many areas of machine learning and statistics, but most methods do not work when there are missing values in the matrix. If it could be done, not only would the observed values be estimated but all the missing values would be predicted. One approach is to impute the missing values, but doing so could distort the observed data due to the sparseness of the original matrix. Another one is to factorize by only using the observed
ratings and try to minimize the squared error.

$$\min \sum_{u,i} (r_{ui} - p_u q_i^T)^2$$  \hspace{1cm} (3)

However, this can result in overfitting the training data. To prevent overfitting a regularization term is introduced to the squared error. Impact of the regularization is controlled by constant $\beta$. [8]

$$\min \sum_{u,i} (r_{ui} - p_u q_i^T)^2 + \beta(||p_u||^2 + ||q_i||^2)$$  \hspace{1cm} (4)

Where $||.||$ denotes the frobenius norm. (4) can be approximated using algorithms explained in 2.4.1 and 2.4.2. This approach has been shown to be very effective and at the same time scalable on very big datasets. One example is the Netflix Prize competition where it was used in the two highest achieving solutions [7]. The factorization can be done beforehand and less memory are used when making a prediction (two vectors of size $n_f$) compared to neighborhood approaches where the whole rating matrix or a subset of it need to be kept in memory.

2.4.1 Stochastic gradient descent

The stochastic gradient descent (SGD) algorithm solves the optimization equation (4). It works by looping through each rating in the training data, tries to predict the rating and calculates a prediction error:

$$e_{ui} = r_{ui} - p_u q_i^T$$  \hspace{1cm} (5)

It then updates the vectors $q_i$ and $p_u$ by a factor proportional to a constant $\alpha$ which we call the learning rate:

$$q_i \leftarrow q_i + \alpha (e_{ui} p_u - \beta q_i)$$  \hspace{1cm} (6)

$$p_u \leftarrow p_u + \alpha (e_{ui} q_i - \beta p_u)$$  \hspace{1cm} (7)

Iteration over the training data continues, calculating the error and updating $q_i$ and $p_u$, until convergence is found or a satisfiable approximation of the original matrix is achieved.

2.4.2 Alternating least squares

Alternating least squares (ALS) [9] is an alternate approach to solving the equation (4). This method functions by taking turns of fixing $p_u$ and $q_i$ and optimizing the other one by solving the least squares problem until convergence is found. The ALS approach is slower compared to the SGD one, but can be parallelized and thus become more effective on certain hardware [10]. Another favourable situation of using ALS is when the matrix $r$ cannot be considered sparse, such as when dealing with implicit datasets [9], where looping through each entry in $r$ such as SGD optimization does would be inefficient.
2.4.3 Matrix factorization with bias

To improve the predictions \[7\] use a bias for movie \(i\), called \(b_i\), bias for user \(u\), called \(b_u\), and global rating average \(\mu\) to model the rating \(r_{ui}\). \(q\) and \(p\) is the same as in equation \[6\].

\[
r_{ui} = \mu + b_i + b_u + p_u q_i^T
\]  

(8)

Here the rating is broken into components where movie and user bias represent the deviation of the movie and user from the global average. Thus \(p_u q_i^T\) becomes only the interaction between the movie and the user, e.g. things the movie and user have in common (the collaborative part). This model modifies the minimized squared error equation \[4\] to include the bias.

\[
\min \sum_{u,i} (r_{ui} - b_i - b_u - p_u q_i^T)^2 + \beta(||p_u||^2 + ||q_i||^2 + b_i^2 + b_u^2) 
\]  

(9)

This equation can be solved using a similar stochastic gradient descent algorithm as \[6\] with addition to learning the bias separately \[5\].

\[
b_i \leftarrow b_i + \alpha(e_{ui} - \beta b_i) 
\]  

(10)

\[
b_u \leftarrow b_u + \alpha(e_{ui} - \beta b_u) 
\]  

(11)

\[
q_i \leftarrow q_i + \alpha(e_{ui} p_u - \beta q_i) 
\]  

(12)

\[
p_u \leftarrow p_u + \alpha(e_{ui} q_i - \beta p_u) 
\]  

(13)

2.5 Relevance and previous research

Matrix factorization techniques can be applied to a number of different systems, not only recommender systems. Understanding the extracted features that are a result of factorization can give deeper insight of the data and also provide use beyond making rating predictions. Previous research \[7\] has shown that using bias can improve the predictive performance of a matrix factorization model. This study will compare this model to a regular decomposition model without the use of bias to understand how they differ and what improvements can be made.
3 Method

This study used matrix factorization to implement a recommender system since it is the current state of the art for implementing collaborative recommender systems. Two implementations of MF was constructed using stochastic gradient descent. SGD was chosen over ALS because we lacked the hardware to make use of the massive parallelization required for ALS to be effective. The SGD algorithms was also easy to implement and efficient enough to run linearly over huge training data consisting of 10 million data points.

3.1 Factorization

The algorithm using stochastic gradient descent was implemented in the following steps:

1. Initialize matrices $p$ and $q$ of size $\text{users} \times K$ and $K \times \text{movies}$ with random values from a uniform distribution over $[0, 1]$ where $K$ stands for the number of features that will be extracted.
2. Iterate over all the observed ratings in training dataset.
   (a) Calculate a predicted rating corresponding to an actual rating.
   (b) Calculate error for the predicted rating.
   (c) Update $p$ and $q$ according to error.

3.2 Parameters

The algorithms are very similar and use the same base, and hence share some parameters determining the effectiveness of the convergence. Tests were made to find the optimal parameters for the base algorithm and were mostly unchanged between the different implementations because of time constraints in testing. Tests used a learning rate $\alpha$ value of 0.005 and regularization constant $\beta$ value of 0.02 for the updating of $p$ and $q$ as described in (6).

3.3 Bias

The factorization algorithm using biases described in section 2.4.3 was implemented using the same steps as the basic algorithm. With changes to step 3 where the new error calculation was used as well as step 4 where updates to $b_i$ and $b_u$ were added according to (10). The predicted ratings was calculated as described in (8).

3.4 MovieLens dataset

The MovieLens datasets [11] was used to test the implemented algorithms. The datasets consist of multiple collections of anonymized data from the MovieLens website [12]. The rating data consist of user id, movie id, a 1-5 rating, and a timestamp. The datasets come in different sizes from 100K ratings up to 22M ratings by 240,000 users on 33,000 movies. The data has been collected in different time periods from 1998 to 2016. In
addition to rating data there are movie data consisting of movie id, title and a list of genres. The choice of using the MovieLens data was made due to the high quality of the data and there being multiple datasets in different sizes.

3.5 Evaluation

3.5.1 Root Mean Squared Error

When evaluating rating predictions a common metric is the Root Mean Squared Error (RMSE). RMSE is a statistical metric that represents the standard deviation between a set of estimated values to the actual values. In recommender systems it has been used to measure how far from the true values a set of predictions was. \[13\]

\[
RMSE = \sqrt{\frac{1}{n} \sum (p_{ui} - r_{ui})^2}
\]

(14)

The RMSE is calculated as follows where \(p_{ui}\) is the predicted rating for user \(u\) and item \(i\), \(r_{ui}\) is the actual rating and \(n\) the number of predictions. This was the evaluation metric used for the Netflix prize [1] and thus a lot of research use this metric, which will give us somewhat comparable results even though we use a different dataset than the Netflix one.

3.5.2 Testing the algorithms

The Root Mean Squared Error was the primary method used to test the accuracy of the prediction algorithms. The latest MovieLens dataset from January 2016 with 100,000 ratings from 700 users on 10,000 movies (100K) was used. The rating data was split 80/20 \% training and testing data. The data was scrambled and split five times with disjoint test sets for five-fold cross-validation of the RMSE. The factorization algorithm was ran on the training set and the RMSE was calculated by testing the predicted rating against the ratings in the test set according to 3.5.1.

The factorization algorithms was also tested against the 10 million ratings dataset from 2009 (10M), consisting of 10,000 movies and 70,000 users. These tests was performed in a similar manner as with 100K. However, cross-validation was not used because of time constraints which means that the RMSE can only be seen as an approximation as it can vary between different test data sets.

3.6 Feature extraction

To analyse how actual feature extraction of the algorithms differ we compare the factorized matrices, more specifically the one representing features for movies named \(q\) in [1]. Each column represents a feature and each row represents a movie. Every entry in the matrix is a value of how well a movie matches the respective feature. To understand what these features corresponds to, we extract the movies which has the highest value of each feature, i.e. the movies best exhibits each feature. Each column
(feature) is sorted in descending order, keeping the original index of the movies, and a number of the first movies are selected creating a list of top movies for each feature. The genres from the MovieLens dataset are also paired with the selected movies for reference. As it turned out, a majority of the top movies for each feature were quite obscure movies which had very few ratings. To filter out the noise we only selected movies that had more than \( n \) ratings. Because the 10M dataset was used for this, \( n \) was set to around 3000 ratings. This also had the benefit of us recognizing the movies which led to easier analysis.

The user-feature matrix called \( p \) (equation (1)) does not have the same value of investigation since the MovieLens dataset is anonymized and does not contain any meta information about the users. Thus, few conclusions could be drawn from this matrix.

### 3.7 Finding similar movies

A common task for a recommendation system, besides finding personalized recommendations for users, is recommending similar items. An example of this can be found in YouTube where each video has a list of similar videos. This can be done effectively using a similar approach to the one in section 3.6 using the movie-feature matrix \( q \). To give a list of recommendations based on an item \( i \) we calculate the cosine similarity between the feature vectors of item \( i \) and the other items in \( q \). Cosine similarity between item \( i \) and \( j \) was calculated as follows:

\[
similarity_{ij} = \frac{q_i \cdot q_j}{||q_i|| \ast ||q_j||}
\]

Where dot indicates the vector dot product and \( ||.|| \) the norm of the vector. This computes the cosine of the angle between the two vectors in an \( K \) dimensional space where \( K \) is the number of features. The similarities are then sorted to give a list of the most similar items. Cosine similarity was chosen as an example for calculating similarities and it proved to work well, but other metrics such as Euclidean distance may work just as well.

### 3.8 Convergence test

Tests were made to see how fast the different algorithms converged to the best RMSE possible. The RMSE and the regularized squared error was calculated after each step. This was done on both the 100K dataset and the 10M dataset. On the bigger dataset this test was made only with 10 features because of time constraints in testing.

### 3.9 Implementation

The algorithms was implemented and tested using Python and the numpy package which contains C extensions for numerical operations and data structures. Matplotlib was used for plotting the data. The tests were run using an Intel i5-3320M 2.6 GHz dual (4 logical) core processor running consistently at the boost clock of 3.1 GHz.
4 Results

We applied our two implementations of matrix factorization, with and without considering item/user bias, on both the 100K and 10M MovieLens datasets. The factorization resulted in decomposing the user-movie rating matrix into two smaller feature matrices, one for users and one for movies. Each user and movie is represented as a vector of features where a feature is represented as a negative or positive value. Features for the movies were analyzed as a whole and per movie basis. Top feature corresponding movies were found for the different features. Cosine similarity for a movie were calculated to find similar movies based on all features. For more detail regarding the implementations see section 3.

4.1 Extracted features

Feature extraction on 10M dataset with 10 features, using bias, achieving an RMSE of 0.823.

<table>
<thead>
<tr>
<th>Feature 7</th>
<th>Feature 9</th>
<th>Feature 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billy Madison Comedy</td>
<td>Lady and the Tramp Animation</td>
<td>Halloween Horror</td>
</tr>
<tr>
<td>Spaceballs Comedy</td>
<td>Dumbo Animation</td>
<td>A Nightmare on Elm Street Horror</td>
</tr>
<tr>
<td>Tommy Boy Comedy</td>
<td>Peter Pan Animation</td>
<td>Scream 2 Horror</td>
</tr>
<tr>
<td>Three Amigos Comedy</td>
<td>Grease Musical</td>
<td>Jaws Horror</td>
</tr>
<tr>
<td>Willow Fantasy</td>
<td>Bambi Animation</td>
<td>The Exorcist Horror</td>
</tr>
</tbody>
</table>

In table 1 we see the top 5 movies in 3 features. Looking at the 3 features we can see that they form 3 distinct categories of movies. Feature 9 consists of animated movies, mostly classic Disney, as well as a musical. Feature 10 consists exclusively of horror movies. Feature 7 consists of comedy movies and a high-fantasy movie.

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>F9</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.27</td>
<td>0.25</td>
<td>-0.09</td>
<td>-0.81</td>
<td>-0.03</td>
<td>0.09</td>
<td>-0.19</td>
<td>-0.28</td>
<td>0.64</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Looking closer at the features of a single movie, Lady and the Tramp has as shown in table 2 a high correspondence to feature 9. What is interesting is that it has a comparable high non-correspondence to feature 4, where the top movies consisted of thrillers. Due to the factorization algorithm does not have non-negativity constraint on the feature values, users can have a negative value for features. Because the movie relevance for a user is calculated using the dot product of the respective feature vectors, suggests that the negative extremes of each feature also creates a category of movies which have less in common with the positive category.
To examine if the negative extremes of each feature also creates movie categories we sort the each feature in ascending order instead of descending and call these features inverted features. The inverted top 5 movies for the example features are shown in table 3.

Table 3: Inverted features top 5

<table>
<thead>
<tr>
<th>Feature 7</th>
<th>Feature 9</th>
<th>Feature 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Crying Game</td>
<td>The Thin Red Line</td>
<td>Moulin Rouge</td>
</tr>
<tr>
<td>Halloween</td>
<td>Fahrenheit 9/11</td>
<td>Documentary</td>
</tr>
<tr>
<td>Thelma &amp; Louise</td>
<td>The Matrix Revolutions</td>
<td>Sci-Fi</td>
</tr>
<tr>
<td>The Exorcist</td>
<td>Dune</td>
<td>Dirty Dancing</td>
</tr>
<tr>
<td>Rosemary’s Baby</td>
<td>The 13th Warrior</td>
<td>Romeo + Juliet</td>
</tr>
<tr>
<td></td>
<td>Action</td>
<td>Romance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mystery Men</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Comedy</td>
</tr>
</tbody>
</table>

Table 3 shows the top 5 movies of the same features as in table 1 but at the negative end of the spectrum. The top movies for the inverted features, like the non-inverted features, creates distinct categories. However, the inverted features is not independent from the non-inverted as shown by the inverted feature 7 that closely resembles non-inverted feature 10, containing two of the same movies and similar genres. Comparing the same features in table 1 and table 3, the top inverted feature movies has little in common with the top non-inverted positive feature movies. Feature 7 consist of comedy movies and inverted feature 7 consist of mostly horror movies. This shows that there exist a clear contrast between the movie categories.

4.2 Recommending similar movies

To examine if the movie-features can be used for finding similar movies we choose an arbitrary movie, Lady and the Tramp, to calculate the similarity to the other movies by comparing their features using cosine similarity.

Table 4: Most similar movies to Lady and the Tramp (1955) using the cosine distance

<table>
<thead>
<tr>
<th>Top 5 similar movies</th>
<th>Cosine distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinderella (1950)</td>
<td>0.027</td>
</tr>
<tr>
<td>Mary Poppins (1964)</td>
<td>0.032</td>
</tr>
<tr>
<td>Peter Pan (1953)</td>
<td>0.032</td>
</tr>
<tr>
<td>Sleeping Beauty (1959)</td>
<td>0.043</td>
</tr>
<tr>
<td>101 Dalmatians (1961)</td>
<td>0.044</td>
</tr>
</tbody>
</table>

Using cosine similarity to calculate similar movies we can see that the system has succeeded in giving similar movies similar features. The movies closes to Lady and the Tramp are all Disney movies from a relatively narrow time period, like Lady and the Tramp. The exception is Mary Poppins which is also a children’s movie from the same time period.
4.3 Comparison of algorithms

This section will compare the two implemented algorithms, a regular MF and MF with ratings modelled with bias, both using SGD. We first investigate how well the algorithms converge towards optimal predictions and how different feature sizes affect the RMSE. The number of features is denoted by $K$.

![Figure 1: Error and RMSE on 100K data set](image)

Figure 1 describes the regularized squared error and RMSE on the 100K dataset as the algorithms progress. Both algorithms continue to minimize the error the longer they are run. The RMSE on the other hand finds a minimum. This indicates overfitting to the training data when running the algorithm for too long. Note that the biased version is considerably better with fewer features compared to the regular MF algorithm that is optimal at 8 features for this data.

The same comparison was performed on the 10M dataset, using 10 features.
Figure 2 describes the regularized squared error and RMSE for the 10M dataset. The curves have similar shapes to those in figure 1, where the error of the biased version also decreased slightly faster. The RMSE takes a lot longer to minimize for the 10M dataset compared to the 100K data set. The regular MF algorithm took 78 steps and the biased one took 82 steps.

Table 5: RMSE performance of algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10M data set</th>
<th>100K data set</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF</td>
<td>0.7999</td>
<td>0.8995</td>
</tr>
<tr>
<td>MF with bias</td>
<td>0.7973 (K=1)</td>
<td>0.8743</td>
</tr>
<tr>
<td>Average rating</td>
<td>1.0430</td>
<td>1.0604</td>
</tr>
</tbody>
</table>

Table 5 shows the best achieved RMSE on both datasets with the two algorithms compared to just using the average rating of all movies to predict ratings. The implemented algorithms perform significantly better on both datasets compared to using the average. The achieved RMSE on the 10M dataset of 0.7973 was better than expected, considering the winner of the Netflix prize challenge reached an RMSE of 0.8567. However, because we are using different datasets and the Netflix one was heavily modified, the results are not directly comparable. In this test using both of the datasets at K=10, MF with bias slightly outperforms the regular MF algorithm. As expected when comparing the results of the different data sets, the performance dramatically improves with more training data.
Table 6: Time per iteration

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10M data set</th>
<th>100K data set</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF</td>
<td>9.2 min</td>
<td>5.2 sec</td>
</tr>
<tr>
<td>MF with bias</td>
<td>12.3 min</td>
<td>8.1 sec</td>
</tr>
</tbody>
</table>

Table 6 describes a dramatic increase in training time to reach the optimal RMSE with the increase in training data.

Figure 3: Time taken

Table 6 and figure 3 describes the time taken when running the algorithms on the hardware described in section 3.9. The bias version on the 10M dataset took roughly 16.8 hours to reach the minimum RMSE (not taking into account the calculation of RMSE as it was done on another core in parallel with the actual factorization). The plain version took in comparison roughly 12 hours to reach the minimum. In 12 hours the MF with bias had reached a better score. We can see from figure 3 that in time taken, the plain MF is faster in getting a decent score but will be outperformed eventually.
5 Discussion

The results show that MF is a powerful approach for implementing recommender systems. Predicting ratings is a critical step when making personalized recommendations and the implemented algorithms predicted the ratings with satisfying accuracy. The RMSE performance of the implemented algorithms corresponds to the performance of the algorithms used in the Netflix prize [1], where factorization models played a big part in the winning contribution. The SGD algorithms used was fairly simple to implement and they learned the models fast on smaller datasets.

The MF approach is not only performant but shows great flexibility as modelling of the ratings can be done differently, improving the performance as showed in the results when introducing bias to the models. Modeling the ratings using biases only slightly improves the RMSE on the 10M dataset but shows an increase in training time. However, on the 100K dataset, the biased version performed considerably better and with a very small number of features. Because the biased version achieved a better RMSE with fewer features, a conclusion can be drawn that the bias captures the user-item interaction better than the feature matrices on the lower quality, smaller dataset.

The MF approach in itself is very scalable, using very little memory to calculate each rating when the model is learned. The SGD algorithm proved to be quick to iterate over small datasets, but when it was used on the 10M dataset, each iteration took a long time and many iterations was required to reach the the optimal RMSE. To use MF in a real world application such as Netflix or Amazon with 100s of millions of data points, an easier to parallelize approach such as ALS may be more practical.

An aim of this study was to understand how the underlying model for MF works by determine what latent features of the decomposed rating matrix represents. Analysing the movie-feature matrix, we can clearly distinguish the difference between the features. The top movies for each feature all consisted of movies from the same or similar genres. From this we draw the conclusion that each feature represent a different movie category where a movie is represented as a combination of the different categories. Because the ratings are calculated as dot product of the user- and movie-feature vectors, the user-feature vector can be seen as a linear combination of the movie-features. This means that we can also interpret the movie-features as representative users that has a distinctive taste, and that each user is a combination of these representatives. However, proving so would require analysing the user-features as well.

The decomposed matrices also proved to be useful beyond predicting ratings. As shown in table [4] the movie-feature matrix can be used to calculate similar movies efficiently and with great accuracy. This study used cosine similarity to calculate the similarity between the movies but other metrics such as Pearson correlation or Euclidean distance could work equally well. Similarly calculation could be done with the user-feature ma-
trix as well, but as we did not have any information about the users, it would be hard to analyse.

5.1 Limitations

A limitation of this study was that we did only run the algorithms on the bigger data set with the number of features set to 10. This was both because of time constraints and because 10 features were a number of features that was easy to get an overview and analyze. Running the algorithms with a different number of features could result in other RMSE scores. It is also unclear what the features extracted would yield with a larger number of features. These might not be as simple as a category of movie genres but instead a more finer grained category.

Due to the lack of data about the users, the extracted features for the users could not be analysed which could otherwise have been of value. This study has only studied matrix factorization within the realms of movie recommendations. Different data might require other models to be effective, such as when no explicit data like ratings is available.

5.2 Further research

MF models show great promise as a general approach for implementing recommender systems. However, this study focused on movie recommendations based on ratings and further research needs to be conducted on other types of data sources to determine if MF is well suited in those cases. New models can be developed for other types of data sources such as content information, implicit feedback and combinations of these with existing models.

Further research could also explore the possibility to extract information from the user-feature matrix to see if the user features correspond equally well with the inherent features of users as the feature-item matrix did. Another area of further research could be making the MF model "on-line", i.e. update the model continuously with new data without having to re-fit the whole model.

5.3 Conclusions

We can conclude that MF is a viable strategy for both making rating predictions and finding similar movies. Introducing bias to the MF model did not have a significant performance increase compared to the one without bias on the bigger dataset but performed considerably better on the smaller dataset. The latent movie-features extracted by MF can be interpreted as movie categories where each movie is represented as a combination of categories. The movie-features can also be used for recommending similar movies.
References
