UNIVERSITEIT • STELLENBOSCH • UNIVERSITY

Movie Recommender System using Matrix Factorization

Lucas Meyer
22614524@sun.ac.za

May 15, 2023

Abstract

This report is a summary of the work that I have done for a MSc course called "Applied Machine Learning at Scale". The course is coordinated by Ulrich Paquet [1] and is presented at the University of Stellenbosch. The main output of the course is to program a recommender system for movies, using the MovieLens dataset. This report is an explanation of the movie recommender system that I have implemented throughout the duration of the course.

1 Introduction

In recent years, recommender systems have gained significant importance in various applications such as e-commerce, social networking, and personalized marketing. Typically, recommender systems assist users in making choices by providing them with personalized recommendations based on their past behavior, preferences, and feedback. In the case of streaming sevices such as Netflix (movies and series), the use of a recommender system may improve user satisfaction and, as a result of this, convince users to continue using the service.

In this report, I present an implementation of a movie recommender system using Yehuda's matrix factorization method [2]. The results of the recommender system is based on the MovieLens dataset [3]. The MovieLens dataset is a commonly used benchmark dataset in the field of recommender systems that contains a collection of user ratings of movies, and genre information relating to each movie in the dataset.

The recommender system discussed in this report utilizes explicit data (user ratings). In contrast, implicit data is observed user behavior such as clicks or purchases (without the explicit rating). The main difference between explicit and implicit movie data is that explicit data may provide more accurate information about user preferences, while implicit data relies on assumptions about user behaviour[1].

The layout of the report is as follows. A description of the MovieLens data and statistical analysis of the dataset(s) is given in Section 2. The matrix factorization method is explained in detail throughout Section 3. The evaluation techniques are briefly discussed in Section 4, and parameter tuning is discussed in Section 5. Section 6 is a summary of several different experimental results, and Section 7 concludes this report with a discussion of the advantages and disadvantages of the recommender system, along with a brief discussion of future work.

[1]Implicit feedback is not utilized in this report. For those interested, Steffen et al. proposed recommender system that utilizes implicit feedback to make predictions using Bayesian analytics [4].

2 Datasets

2.1 Description of the MovieLens Datasets

There are many versions of the MovieLens dataset. Each version of the MovieLens dataset contains a collection of movie ratings that range from 0.5 to 5, with increments of 0.5. The datasets also provide additional metadata about the movies (title, year of release, and genre). There are two MovieLens datasets that are used in this report:

1. **Latest-small dataset:** This dataset contains $100,000$ ratings from 610 different users for 9724 unique movies. This dataset is a subset of the full MovieLens dataset and is commonly used for quick experimentation and prototyping of recommender systems.
2. **25M dataset:** This dataset contains over 25 million ratings from $162,541$ users for $59,047$ unique movies. This dataset is suitable for training and testing large-scale recommender systems and is commonly used in academic research and industry applications.

For the remainder of the report, I refer to the "Latest-small" dataset as the *small* dataset and the "25M" dataset as the *full* dataset.

2.2 Statistical analysis of dataset

In the field of machine learning, it is important to analyze the data which you are using to build your model. Data quality issues may bias the recommender system, resulting in undesirable recommendations. Most of these data quality issues can be mitigated by using different preprocessing techniques.

2.2.1 Missing values

For the full dataset, some movie IDs that appear in the `movies.csv` file do not appear in the `ratings.csv` file (some movies in the dataset are not rated). The reason for this is that the full dataset has been modified so that every user has rated at least 20 movies. This is why some movies do not have ratings. This issue is solved by removing all movie IDs in the `movies.csv` file that does not appear in the `ratings.csv` file.

2.2.2 Scale-free properties

A scale-free distribution of ratings is an essential characteristic for a recommender system. Recommender systems require this property to more accurately recommend popular movies since they are more likely to have a sufficient number of ratings to make meaningful predictions. Conversely, if all movies had the same number of ratings, it would be challenging to differentiate which movies are more popular or preferred by users. Therefore, a scale-free distribution of ratings allows the recommender system to identify and recommend movies that are more likely to be appreciated by users, based on the preference patterns observed in the dataset.

It is possible to identify whether a data distribution is scale-free based on whether the data distribution has a power law. A simple approach to identify whether a power law exists for the MovieLens ratings is to plot the distributions of ratings per user and ratings per item (movie) using a logarithmic scale on both axis. Figure 1 (on the next page) shows the distribution of the small dataset ratings (left) and full dataset ratings (right), where items are represented in blue and users are represented in green. Note that both datasets have been preprocessed so that each user has rated at least 20 movies.

A logarithmic scale is applied to each dataset distribution to emphasize whether the distributions are power-law distributions or not. In both Figures, the rating distributions suggests that there are power laws in the data, which implies that the data is scale free.

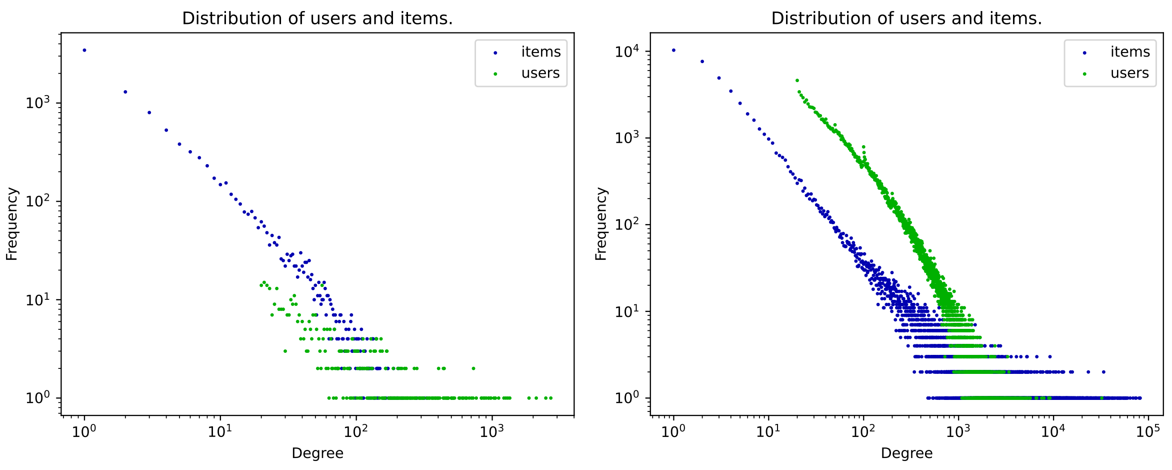

Figure 1: The distribution of ratings per user and ratings per item for the small dataset (left) and the full dataset (right), using a logarithmic scale on both axis.

2.2.3 Distribution of rating values

Figure 2 and 3 shows the number of ratings for each possible rating value from 0.5 to 5 for the small and full datasets, respectively. Figure 4 and 5 shows the number of ratings for the integer converted ratings (1 to 5) for the small and full datasets, respectively.

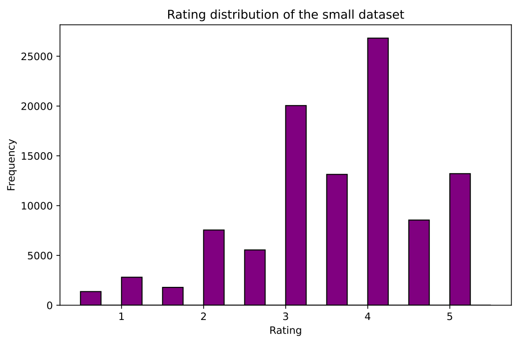

Figure 2: The number of ratings for each possible rating value from 0.5 to 5 for the small dataset.

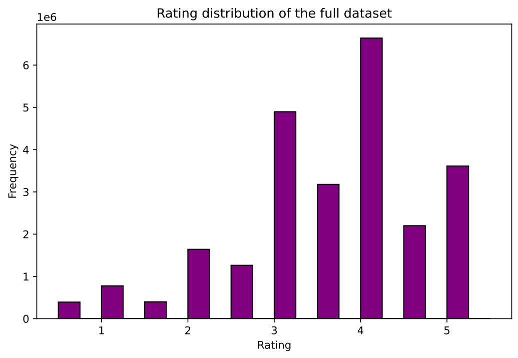

Figure 3: The number of ratings for each possible rating value from 0.5 to 5 for the full dataset.

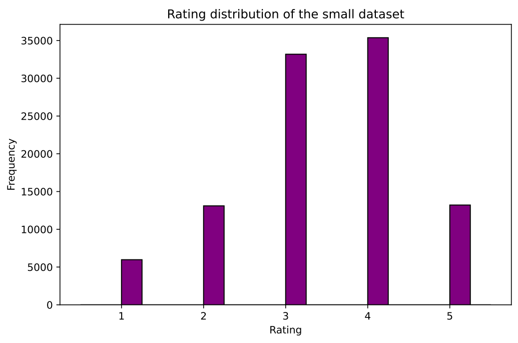

Figure 4: The number of ratings for the integer converted ratings (1 to 5) for the small dataset.

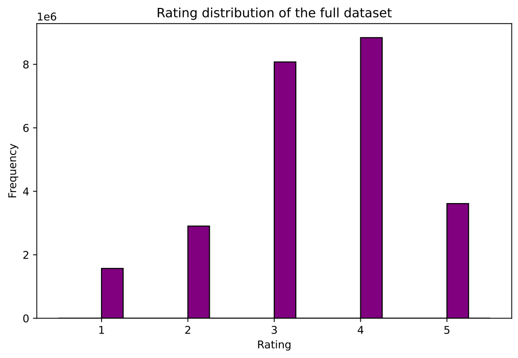

Figure 5: The number of ratings for the integer converted ratings (1 to 5) for the full dataset.

3 Yehuda's matrix factorization method

Yehuda's matrix factorization method is a popular approach to *collaborative filtering* in recommender systems. Collaborative filtering is a technique in which movies that one user enjoys will be recommended to another user that watches similar movies. Thus, predictions are based on the similarity of movies that a user watches to another user, and not the content of the movie itself.

3.1 Trait vector representation and the factorization method

A common technique in machine learning is to transform the input data into some sort of feature representation. Typically, this is done to model the input data according to some probability distribution. For the factorization method, we assume that each user and each item is represented as a k-dimensional trait vector. The factorization method works as follows:

- Each user $m \in \{1, \dots, M\}$ in the dataset is represented as a trait vector $\mathbf{u}_m \in \mathcal{R}^k$.
- Each item $n \in \{1, \dots, N\}$ in the dataset is represented as a trait vector $\mathbf{v}_n \in \mathcal{R}^k$.
- The user vectors $\{\mathbf{u}_0, \dots, \mathbf{u}_M\}$ are combined row-wise into a $M \times k$ matrix, denoted by U.
- The item vectors $\{\mathbf{v}_0, \dots, \mathbf{v}_N\}$ are combined row-wise into a $N \times k$ matrix, denoted by V.
- Following from this, the method factorizes the *user-to-item rating* matrix, R, into lower-rank matrices U and V^T such that:

$$R \approx UV^T. \tag{1}$$

The idea of the factorization method is to model the dot product between a user vector $\mathbf{u}_m$ and an item vector $\mathbf{v}_n$ by the random variable r_{mn}, which is the predicted rating of a user m for a movie n. This method allows us to predict the user rating for a movie that the user has not rated.

3.2 Initialization and further motivation

Each user and item trait vector is initialized by sampling k points from a Gaussian distribution with probability density function $\sim \mathcal{N}(\mu = \sqrt{2.5/k},\, \sigma^2 = \lambda)$, where k is a control parameter and λ is a hyperparameter. The reason for initializing the trait vectors in this way is to ensure that the inner product between a user vector $\mathbf{u}_m$ and an item vector $\mathbf{v}_n$ (which is r_{mn}) is *most likely* between 0 and 5, for any value of $k \leq 1$. This notion is further motivated by Figures 6 and 7.

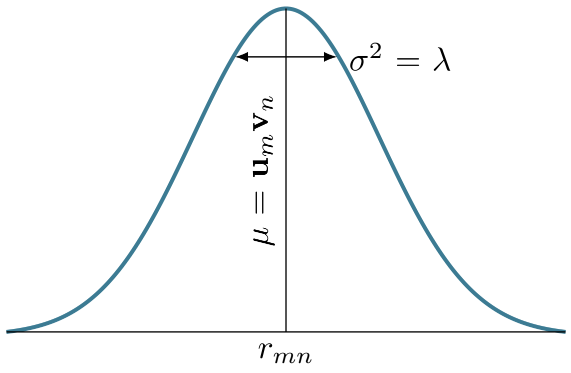

Figure 6: A diagram of the Gaussian (normal) distribution that describes the predicted rating r_{mn}, where $\mathbf{u}_m$ is the trait vector of user m, and $\mathbf{v}_n$ is the trait vector of item n.

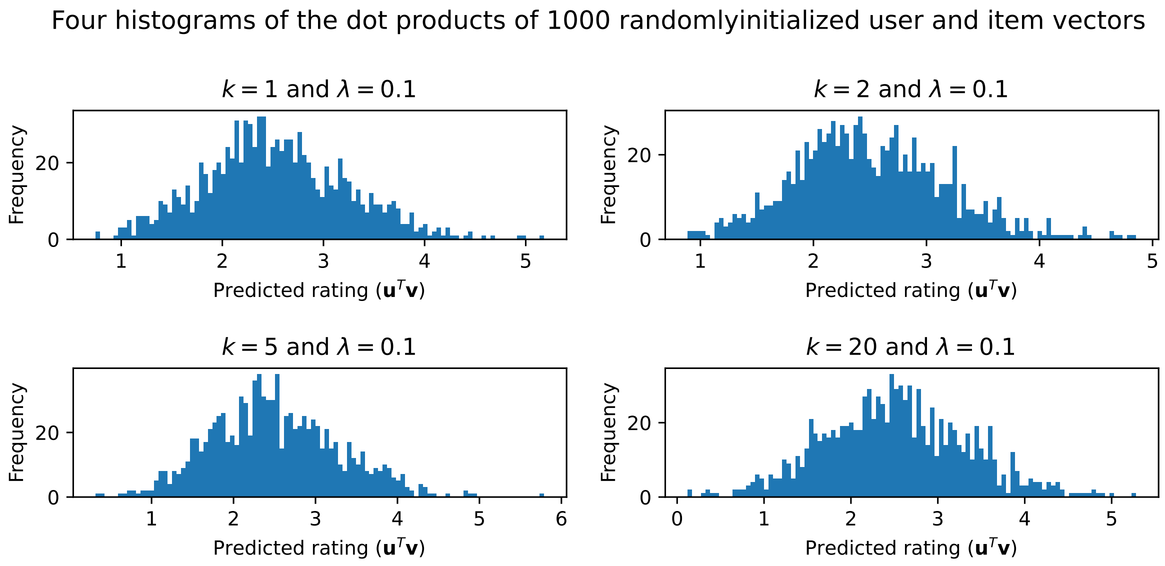

Figure 7: Each histogram represent a sample of 1000 inner products of randomly initialized user vectors ($\mathbf{u}$) and item vectors ($\mathbf{v}$). The dimensionality of the trait vectors is $k \in \{1, 2, 5, 20\}$ and the value of λ is set to 0.1 for all four histogram plots. As you may notice, the value of the inner products (predicted ratings) for all four plots are mostly between 0 and 5, irregardless of the value of k.

3.2.1 Embedding in latent space

The latent space refers to the abstract space that is learned by the recommender system when it is trained using the ALS algorithm. Ideally, similar users and items should be embedded in the same region of the latent space, which results in a large inner product of a similar user and item (see Figure 8). Conversely, dissimiliar users and items should be embedded in different regions of the latent space, which results in a small inner product of a dissimilar user and item.

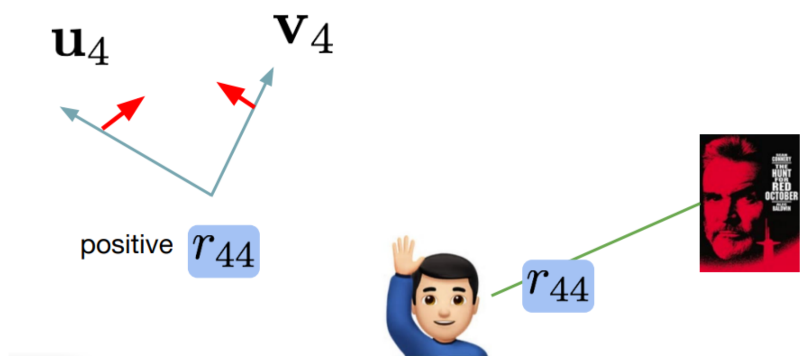

Figure 8: A diagram of what happens during learning, where similar user vectors and item vectors are embedded in the same direction in the latent space ($k = 2$).
This diagram was created by Ulrich Paquet [1].

3.3 ALS: Using only trait vectors

The alternating least squares (ALS) algorithm is used to obtain the matrix factorization. The ALS algorithm iterates between solving for U and V until convergence. More information on the ALS algorithm can be found here.

The loss function that is used for optimization is the log-likelihood function of all the user and item vectors. A regularization term is also added for the user and item vectors, where the regularization coefficient is $\tau/2$. The regularized log-likelihood function is given by:

$$\begin{aligned}\mathcal{L} &= \log p(R|U,V) + \log p(U) + \log p(V) \\ &= -\frac{\lambda}{2}\sum_{mn}(r_{mn} - \mathbf{u}_m^T\mathbf{v}_n)^2 - \frac{\tau}{2}\sum_m \mathbf{u}_m^T\mathbf{u}_m - \frac{\tau}{2}\sum_n \mathbf{v}_n^T\mathbf{v}_n + \text{const.}\end{aligned} \tag{2}$$

3.3.1 Update equation for $\mathbf{u}_m$

The partial derivative of the loss function (Equation 2) with respect to $\mathbf{u}_m$ is written as follows, where $\Omega(m)$ is the set of movies that user m has rated:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = -\lambda \sum_{n\in\Omega(m)} -\mathbf{v}_n(r_{mn} - \mathbf{u}_m^T\mathbf{v}_n) - \tau\mathbf{u}_m.$$

Setting $\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m}$ equal to zero and rearranging terms leads to

$$\left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n^T\mathbf{v}_n + \tau\mathbf{I}\right)\mathbf{u}_m = \lambda \sum_{n\in\Omega(m)} \mathbf{v}_n r_{mn}.$$

Multiplying both sides of the equation above by the inverse of the matrix on the left-hand side leads to the update equation for $\mathbf{u}_m$:

$$\mathbf{u}_m = \left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n^T\mathbf{v}_n + \tau\mathbf{I}\right)^{-1}\left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n r_{mn}\right). \tag{3}$$

3.3.2 Update equation for $\mathbf{v}_n$

The partial derivative of the loss function (Equation 2) with respect to $\mathbf{v}_n$ is written as follows, where $\Psi(n)$ is the set of users that have rated movie n:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_n} = -\lambda \sum_{m\in\Psi(n)} -\mathbf{u}_m(r_{mn} - \mathbf{u}_m^T\mathbf{v}_n) - \tau\mathbf{v}_n.$$

Setting $\frac{\partial \mathcal{L}}{\partial \mathbf{v}_n}$ equal to zero and rearranging terms:

$$\left(\lambda \sum_{m\in\Psi(n)} \mathbf{u}_m^T\mathbf{u}_m + \tau\mathbf{I}\right)\mathbf{v}_n = \lambda \sum_{m\in\Psi(n)} \mathbf{u}_m r_{mn}.$$

Multiplying both sides of the equation above by the inverse of the matrix on the left-hand side leads to the update equation for $\mathbf{v}_n$:

$$\mathbf{v}_n = \left(\lambda \sum_{m\in\Psi(n)} \mathbf{u}_m^T\mathbf{u}_m + \tau\mathbf{I}\right)^{-1}\left(\lambda \sum_{m\in\Psi(n)} \mathbf{u}_m r_{mn}\right). \tag{4}$$

3.4 ALS: Adding bias terms

Now bias terms are added to the likelihood function, one for each user and one for each item. A regularization term is also added for each bias term, where the regularization coefficient is $-\alpha/2$ (and α is a hyperparameter). The bias terms also need to be updated per iteration. The regularized log-likelihood function with bias terms included is given by:

$$\begin{aligned}\mathcal{L} = &-\frac{\lambda}{2}\sum_{mn}\left(r_{mn} - (\mathbf{u}_m^T\mathbf{v}_n + b_m^{(u)} + b_n^{(v)})\right)^2 \\ &-\frac{\tau}{2}\sum_m \mathbf{u}_m^T\mathbf{u}_m - \frac{\tau}{2}\sum_n \mathbf{v}_n^T\mathbf{v}_n \\ &-\frac{\alpha}{2}\sum_m (b_m^{(u)})^2 - \frac{\alpha}{2}\sum_n (b_n^{(v)})^2 + \text{const.}\end{aligned} \tag{5}$$

3.4.1 Update equations for $\mathbf{u}_m$ and $\mathbf{v}_n$

The update equations for $\mathbf{u}_m$ and $\mathbf{v}_n$ with the bias terms is very similar to Equations 3 and 4:

$$\mathbf{u}_m = \left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n^T\mathbf{v}_n + \tau\mathbf{I}\right)^{-1}\left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n(r_{mn} - b_m^{(u)} - b_n^{(v)})\right), \tag{6}$$

$$\mathbf{v}_n = \left(\lambda \sum_{m\in\Psi(n)} \mathbf{u}_m^T\mathbf{u}_m + \tau\mathbf{I}\right)^{-1}\left(\lambda \sum_{m\in\Psi(n)} \mathbf{u}_m(r_{mn} - b_m^{(u)} - b_n^{(v)})\right). \tag{7}$$

3.4.2 Update equation for $b_m^{(u)}$

The partial derivative of the loss function (Equation 5) with respect to $b_m^{(u)}$ is written as:

$$\frac{\partial\mathcal{L}}{\partial b_m^{(u)}} = -\lambda \sum_{n\in\Omega(m)} -(r_{mn} - \mathbf{u}_m^T\mathbf{v}_n - b_m^{(u)} - b_n^{(v)}) - \alpha b_m^{(u)}.$$

Setting $\frac{\partial\mathcal{L}}{\partial b_m^{(u)}}$ equal to zero and rearranging terms leads to the update equation for $b_m^{(u)}$:

$$\alpha b_m^{(u)} + \lambda|\Omega(m)|b_m^{(u)} = \lambda \sum_{n\in\Omega(m)} (r_{mn} - \mathbf{u}_m^T\mathbf{v}_n - b_n^{(v)})$$

$$b_m^{(u)} = \frac{\lambda}{\alpha + \lambda|\Omega(m)|}\sum_{n\in\Omega(m)} (r_{mn} - \mathbf{u}_m^T\mathbf{v}_n - b_n^{(v)}). \tag{8}$$

3.4.3 Update equation for $b_n^{(v)}$

The partial derivative of the loss function (Equation 5) with respect to $b_n^{(v)}$ is written as:

$$\frac{\partial\mathcal{L}}{\partial b_n^{(v)}} = -\lambda \sum_{m\in\Psi(n)} -(r_{mn} - \mathbf{v}_n^T\mathbf{u}_m - b_m^{(u)} - b_n^{(v)}) - \alpha b_n^{(v)}.$$

Setting $\frac{\partial\mathcal{L}}{\partial b_n^{(v)}}$ equal to zero and rearranging terms leads to the update equation for $b_n^{(v)}$:

$$\alpha b_n^{(v)} + \lambda|\Psi(n)|b_n^{(v)} = \lambda \sum_{m\in\Psi(n)} (r_{mn} - \mathbf{u}_m^T\mathbf{v}_n - b_m^{(u)})$$

$$b_n^{(v)} = \frac{\lambda}{\alpha + \lambda|\Psi(n)|}\sum_{m\in\Psi(n)} (r_{mn} - \mathbf{u}_m^T\mathbf{v}_n - b_m^{(u)}). \tag{9}$$

3.5 ALS: Using genre metadata

It is possible to use genre metadate for each movie to further expand the log-likelihood function. This is done by learning genre trait vectors ($\mathbf{f}_i$) and modifying the regularizer for the item vectors. This causes the movies to be embedded into more distinct areas of the latent space, depending on the genres associated with each movie. A regularization term is introduced for the genre vectors ($\mathbf{f}_i$), where the regularization coefficient is $-\mathcal{B}/2$ (and $\mathcal{B}$ is a hyperparameter). The regularized log-likelihood function with biases and genre metadata is written as follows, where $\mathcal{X}(n)$ is the set of genres that are associated with movie n:

$$\begin{aligned}\mathcal{L} = &-\frac{\lambda}{2}\sum_{mn}\left(r_{mn} - (\mathbf{u}_m^T\mathbf{v}_n + b_m^{(u)} + b_n^{(v)})\right)^2 \\ &-\frac{\tau}{2}\sum_m \mathbf{u}_m^T\mathbf{u}_m - \frac{\tau}{2}\sum_n(\mathbf{v}_n - \frac{1}{\sqrt{|\mathcal{X}(n)|}}\sum_{i\in\mathcal{X}(n)}\mathbf{f}_i)^T(\mathbf{v}_n - \frac{1}{\sqrt{|\mathcal{X}(n)|}}\sum_{i\in\mathcal{X}(n)}\mathbf{f}_i) \\ &-\frac{\alpha}{2}\sum_m(b_m^{(u)})^2 - \frac{\alpha}{2}\sum_n(b_n^{(v)})^2 - \frac{\mathcal{B}}{2}\sum_i \mathbf{f}_i^T\mathbf{f}_i + \text{const.}\end{aligned} \tag{10}$$

The update equations for $\mathbf{u}_m$, $b_m^{(u)}$, and $b_n^{(v)}$ remain the same as in Section 3.4.

3.5.1 Update equation for $\mathbf{v}_n$

Using genre metadata changes the update equation for $\mathbf{v}_n$:

$$\mathbf{v}_n = \left(\lambda\sum_{m\in\Psi(n)}\mathbf{u}_m^T\mathbf{u}_m + \tau\mathbf{I}\right)^{-1}\left(\lambda\sum_{m\in\Psi(n)}\mathbf{u}_m(r_{mn} - b_m^{(u)} - b_n^{(v)}) + \frac{\tau}{\sqrt{|\mathcal{X}(n)|}}\sum_{i\in\mathcal{X}(n)}\mathbf{f}_i\right). \tag{11}$$

3.5.2 Update equation for $\mathbf{f}_i$

The partial derivative of the loss function (Equation 10) with respect to $\mathbf{f}_i$ is written as follows, where $\mathcal{Z}(i) = \{n : i \in \mathcal{X}(n)\}$ (all movies that contain the genre i):

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\mathbf{f}_i} &= \tau\left[\sum_{\mathcal{Z}(i)}\left(\mathbf{v}_n - \frac{1}{\sqrt{|\mathcal{X}(n)|}}\sum_{j\in\mathcal{X}(n)}\mathbf{f}_j\right)\right]\left(\sum_{\mathcal{Z}(i)}\frac{1}{\sqrt{|\mathcal{X}(n)|}}\right) - \mathcal{B}\mathbf{f}_i \\ &= \tau\left[\sum_{\mathcal{Z}(i)}\left(\mathbf{v}_n - \frac{1}{\sqrt{|\mathcal{X}(n)|}}\sum_{\{j\in\mathcal{X}(n)\}/i}\mathbf{f}_j\right) - \left(\sum_{\mathcal{Z}(i)}\frac{1}{\sqrt{|\mathcal{X}(n)|}}\right)\mathbf{f}_i\right]\left(\sum_{\mathcal{Z}(i)}\frac{1}{\sqrt{|\mathcal{X}(n)|}}\right) - \mathcal{B}\mathbf{f}_i.\end{aligned} \tag{12}$$

Setting $\dfrac{\partial\mathcal{L}}{\partial\mathbf{f}_i}$ equal to zero and rearranging terms:

$$\tau\left(\sum_{\mathcal{Z}(i)}\frac{1}{\sqrt{|\mathcal{X}(n)|}}\right)^2\mathbf{f}_i + \mathcal{B}\mathbf{f}_i = \tau\left(\sum_{\mathcal{Z}(i)}\frac{1}{\sqrt{|\mathcal{X}(n)|}}\right)\left[\sum_{\mathcal{Z}(i)}\left(\mathbf{v}_n - \frac{1}{\sqrt{|\mathcal{X}(n)|}}\sum_{\{j\in\mathcal{X}(n)\}/i}\mathbf{f}_j\right)\right].$$

Simplifying the equation above yields the update equation for $\mathbf{f}_i$:

$$\mathbf{f}_i = \frac{\tau\left(\sum_{\mathcal{Z}(i)}\frac{1}{\sqrt{|\mathcal{X}(n)|}}\right)}{\tau\left(\sum_{\mathcal{Z}(i)}\frac{1}{\sqrt{|\mathcal{X}(n)|}}\right)^2 + \mathcal{B}}\left[\sum_{\mathcal{Z}(i)}\left(\mathbf{v}_n - \frac{1}{\sqrt{|\mathcal{X}(n)|}}\sum_{\{j\in\mathcal{X}(n)\}/i}\mathbf{f}_j\right)\right]. \tag{13}$$

3.6 Parallelized ALS algorithm

Yehuda's factorization method is convenient to use, since it is easy to parallelize. Note that the update equation of one user trait vector is independent from the update equation of a different user trait vector. The same holds for the item/genre vector and the bias terms.

This means that the ranges $0, \dots, M$ and $0, \dots, N$ can be split up into the number of CPU cores of the computer the recommender system is being trained. I implemented multiprocessing with four cores, and the parallel version of my code executed significantly faster than my unparallelized code.

4 Evaluation techniques

The loss function (regularized log-likelihood) should increase monotonically over time. There are also other techniques to evaluate performance of the model, which are explained in the following subsections.

4.1 Root mean squared error

The root mean squared error (RMSE) is computed by averaging the squared errors over all user item pairs, $(m, n) \in \mathcal{R}$:

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{R}|} \sum_{(m,n)\in\mathcal{R}} (\hat{r}_{mn} - r_{mn})^2}. \tag{14}$$

The RMSE metric is commonly used in machine learning algorthms. Note that there are issues associated with this metric. Firstly, the RMSE metric tends to emphasize large errors rather than preserving fine neighbourhoods. Secondly, RMSE tends to emphasize popular items that most users have rated.

4.2 Mean absolute error

The mean absolute error (MAE) is computed by averaging the absolute errors over all user item pairs, $(m, n) \in \mathcal{R}$:

$$\text{MAE} = \frac{1}{|\mathcal{R}|} \sum_{(m,n)\in\mathcal{R}} |\hat{r}_{mn} - r_{mn}|. \tag{15}$$

Unlike the RMSE, the MAE metric does not emphasize large errors. However, in machine learning it is less commonly used than the RMSE metric, due to the difficulties working with the derivative of the MAE function.

4.3 Evaluation by inspection

A simple way to check whether a recommender system is working properly is to see create a "dummy user" that has rated a collection of movies. Now we can use the trained recommender system to predict movies for the dummy user. Using human intuition, we can argue that if the top recommendations are movies that are similar to the movies that the dummy user has rated highly, then we assume that the recommender system is working. Conversely, if the recommender system recommends a Horror movie for a dummy user that rated a Children movie, then we assume something is wrong with the recommender system.

Note that while this technique is simple and intuitive, there are several issues associated with this technique, such as:

- **Cold start problem**: Regardless of what the dummy user has rated, the same movies are recommended - usually movies that are very popular (or "polarizing movies"). Fortunately, the use of genre trait vectors in the ALS algorthm mitigates this problem.
- **Outliers**: There could be two movies, with two completely different genres, that are rated by a lot of the same users. This may cause unexpected recommendations. This is one of the disadvantages of collaborative filtering, however, the use of genre trait vectors slightly mitigates this problem.

5 Parameter tuning

Due to time constraints, I did not tune any parameters for the recommender system. The first subsection discusses how I would have tuned the control parameter and hyperparameters, and the second subsection discusses what values I selected for the parameters and hyperparameters, and also why I selected those values.

5.1 Parameter tuning using cross-validation

There is one control parameter, k, and four hyperparameters: $\lambda, \tau, \alpha, \beta$. Parameter tuning is best performed using a random cross-validation search. Random search can explore a wider range of parameter values and potentially find a better set of parameters than with a grid search approach. To tune the control parameter and four hyperparameters using random cross-validation search:

1. Define the range of values for each parameter.
2. Divide the small/full dataset into a training and validation set.
3. Define the number of iterations and the number of folds.
4. For each iteration, select random values for the parameters from their defined ranges.
5. Train the model with the selected parameter values, and evaluate the performance of the model on the validation set.
6. Repeat steps 4 and 5 for the specified number of iterations and folds.
7. Select the combination of parameter values that maximizes the performance of the model on the validation set.

5.2 Choosing parameter values

Since no parameter tuning was implemented, I chose the value for the control parameter and hyperparameters based on my intuition of the ALS algorithm. The latent space dimensionality (k) is chosen as 20, recommended by Ulrich Paquet. A value of $k << 20$ will result in a low-dimensional feature representation of the user and item data, which means the ALS algorithm may underfit the data. A value of $k >> 20$ will cause a non-linear increase in computational complexity of the ALS algorithm. Since I (regrettably) implemented my recommender system using `Python`, my computer takes too much time to run the ALS algorithm with a value of $k >> 20$.

The value of λ is chosen as 0.1, and Figure 7 is the main reason for this choice. The other hyperparameters, which are τ, α, and β, are coefficients for the penalty terms of the user, item, and genre trait vectors. I set all three of these hyperparameters equal to 0.01. If the value for β is decreased, the genre features vectors are less penalized in the loss function, which increases the content-based filtering of the recommender system.

The control parameter and hyperparameter values were chosen as follows:

$$k = 20;\ \lambda = 0.1;\ \tau = 0.01;\ \alpha = 0.01;\ \beta = 0.01. \tag{16}$$

6 Experimenal results

Several different experimental results were gathered by training a model using the ALS algorithm. In most cases, the models were trained using the ALS update equations described in Section 3.5, and the values for the control parameter and hyperparameters were selected according to Equation 16.

6.1 Loss and error results

6.1.1 Regularized log-likelihood over time

Figures 9 and 10 show the value of the loss function (Equation 10) for 25 iterations of the ALS algorithm. Note that the value of the loss function is for the training set and is monotonically increasing (since the loss function is negative).

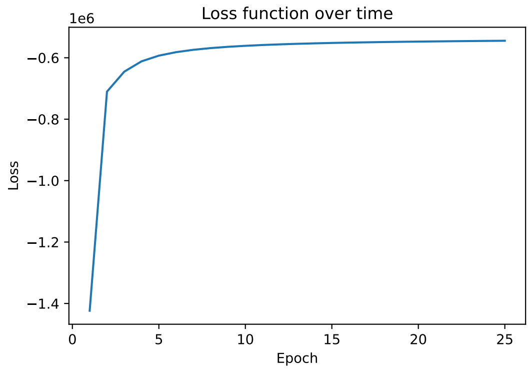

Figure 9: Plot of the loss function values over time (using ALS with biases and trait vectors) for the small dataset.

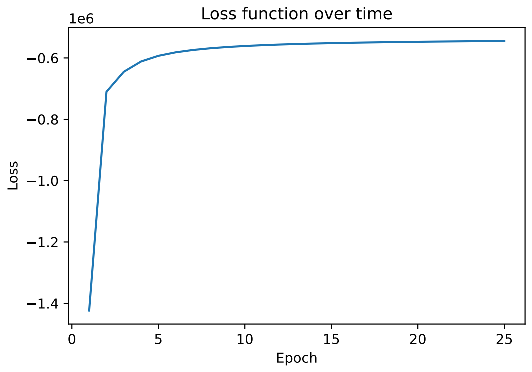

Figure 10: Plot of the loss function values over time (using ALS with biases and trait vectors) for the full dataset.

6.1.2 RMSE and MAE over time

Figures 11 and 12 show the value of the RMSE and MAE (Equations 14 and 15) for 25 iterations of the ALS algorithm. Note that ideally both the RMSE and MAE value should decrease over time.

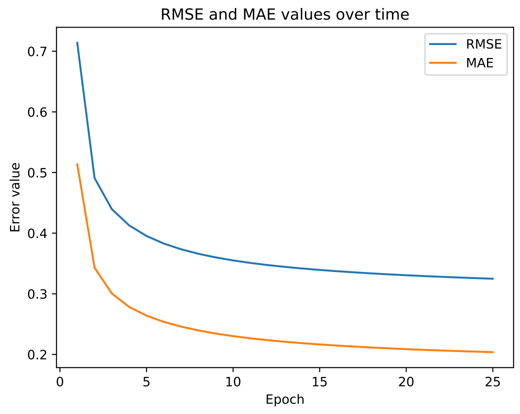

Figure 11: Plot of the RMSE and MAE values over time (using ALS with biases and trait vectors) for the small dataset.

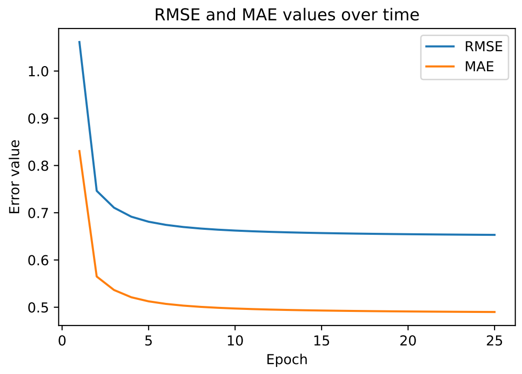

Figure 12: Plot of the RMSE and MAE values (using ALS with biases and trait vectors) for the full dataset.

6.2 Two-dimensional trait vector embeddings

6.2.1 Item vector embeddings

Figure 13 and 14 show a selection of movies and their two-dimensional embeddings. For both Figures, the ALS alorithm was run for 10 iterations ($k = 2$), and the hyperparameters are selected as before. This is done to see whether similar movies are embedded together and opposite movies are embedded in "opposite" parts of the space.

For both datasets, the trait vectors in blue indicate that childrens' movies (such as "Despicable Me" and "Shrek") are embedded far away from horror movies (such as "Saw" and "The Conjuring").

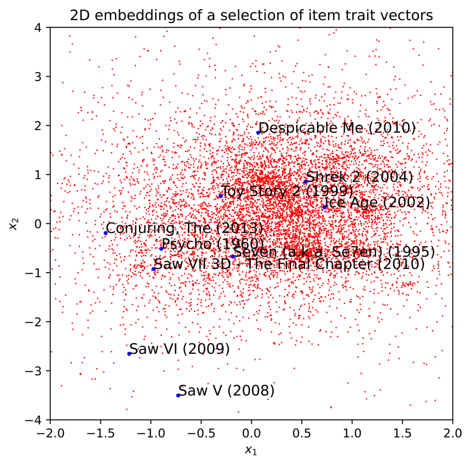

Figure 13: A scatter plot of the item trait vectors for the small dataset.

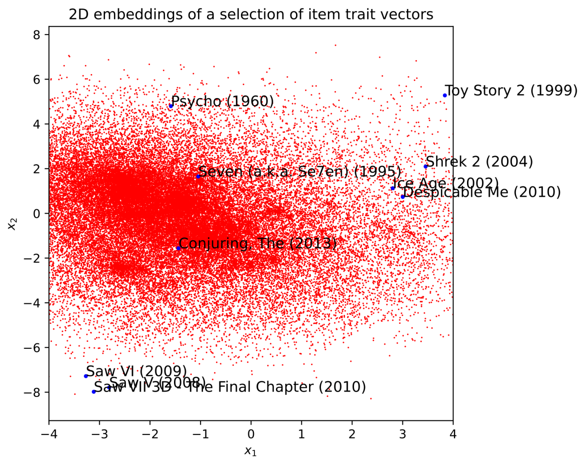

Figure 14: A scatter plot of the item trait vectors for the full dataset.

6.2.2 Genre vector embeddings

Figure 15 and 16 the two-dimensional embeddings of the genre trait features. For both Figures, the ALS algorithm was run for 10 iterations ($k = 2$), and the hyperparameters are selected as before. This is done to see whether similar genres are embedded together and opposite genres are embedded in "opposite" parts of the space.

For both datasets, the genre trait vectors for "Children" and "Horror" are embedded in opposite parts of the space. Additionally, the trait vectors for the genres "Crime", "Mystery", and "Thriller" are relatively close to each other, which makes sense if you think about how movies are categorized by all three of these genres simultaneously.

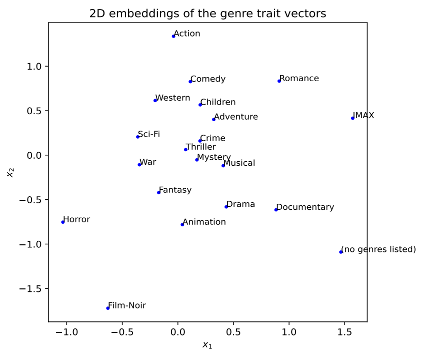

Figure 15: A scatter plot of the genre trait vectors for the small dataset.

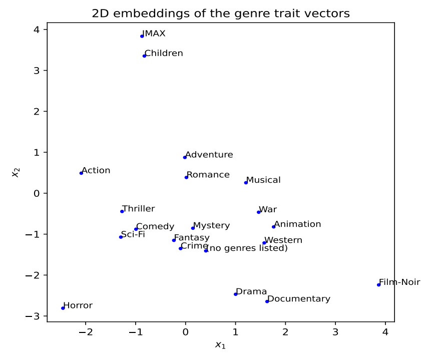

Figure 16: A scatter plot of the genre trait vectors for the full dataset.

6.3 Dummy user predictions

For this section of results, we set up "dummy users" to show that if a the dummy user liked movie X (handpicked movie choice) – the recommendations Y make sense. This is done by creating a user that has rated one movie 5 stars, updating the user vector and user bias for a number of iterations, and returning the ten largest predicted ratings.

6.3.1 Small dataset results

The results (Tables 1 - 3) shown are for three dummy users using the small dataset. The results show promise, however the small dataset contains too few movies and user ratings to get desirable recommendations.

The recommendations for the "Star Wars" movie are decent, but I would expect more Star Wars movies to be recommended. The recommendations for the "Lord of the Rings" movie are the best out of the three, since the entire Lord of the Rings trilogy is recommended within the top 10 recommendations. Finally, the recommendations for the "Toy Story" movie are the worst, because the system did not recommend a Toy Story movie and it also only recommended four other animation movies.

Table 1: Predictions for Star Wars: Episode II - Attack of the Clones (2002)

Rank	**Movie**	**Rating**	**Genres**
1	**Star Wars: Episode II - Attack of the Clones (2002)**	1.9916	Action,Adventure,Sci-Fi
2	Matrix Revolutions, The (2003)	1.9446	Action,Adventure,Sci-Fi,Thriller
3	Hellboy (2004)	1.8965	Action,Adventure,Fantasy,Horror
4	Sgt. Bilko (1996)	1.8923	Comedy
5	American Pie 2 (2001)	1.8667	Comedy
6	Oblivion (2013)	1.8527	Action,Adventure,Sci-Fi,IMAX
7	We Were Soldiers (2002)	1.8372	Action,Drama,War
8	Saw (2004)	1.8041	Horror,Mystery,Thriller
9	Click (2006)	1.7917	Adventure,Comedy,Drama
10	Boys on the Side (1995)	1.7812	Comedy,Drama

Table 2: Predictions for Lord of the Rings: The Return of the King (2003)

Rank	**Title**	**Score**	**Genres**
1	Old School (2003)	1.1514	Comedy
2	In the Name of the Father (1993)	1.1041	Drama
3	**Lord of the Rings: Two Towers, The (2002)**	1.1007	Adventure, Fantasy
4	**Lord of the Rings: Fellowship of the Ring, The (2001)**	1.0999	Adventure, Fantasy
5	English Patient, The (1996)	1.0978	Drama, Romance, War
6	Boys Don't Cry (1999)	1.0780	Drama
7	Willow (1988)	1.0754	Action, Adventure, Fantasy
8	**Lord of the Rings: Return of the King, The (2003)**	1.0752	Action, Adventure, Drama, Fantasy
9	West Side Story (1961)	1.0709	Drama, Musical, Romance
10	Hellboy (2004)	1.0420	Action, Adventure, Fantasy, Horror

Table 3: Predictions for Toy Story (1995).

Rank	**Title**	**Rating**	**Genres**
1	Wild Things (1998)	1.4163	Crime, Drama, Mystery, Thriller
2	Bend It Like Beckham (2002)	1.4084	Comedy, Drama, Romance
3	River Runs Through It, A (1992)	1.3917	Drama
4	Wallace & Gromit: The Best of Aardman Anim...	1.3789	Adventure, Animation, Comedy
5	Lost Boys, The (1987)	1.3717	Comedy, Horror, Thriller
6	Iron Giant, The (1999)	1.3523	Adventure, Animation, Children, Drama, Sci-Fi
7	Hunchback of Notre Dame, The (1996)	1.3457	Animation, Children, Drama, Musical, Romance
8	Jackass: The Movie (2002)	1.3447	Action, Comedy, Documentary
9	Crank (2006)	1.3264	Action,Thriller
10	Gone with the Wind (1939)	1.3259	Drama,Romance,War

6.3.2 Full dataset results

The same three movies was used to create dummy users for the full dataset. The results for the three dummy users, summarized by Tables 4 - 6, are much better and consistent than for the small dataset.

The recommendations for the Star Wars movie are great, with four Star Wars movies in the top 8 recommendations. The recommendations for the Lord of the Rings movie are again the best out of the three movies, since the Lord of the Rings and "Hobbit" trilogies (6 movies) are in the top 10 recommendations. Finally, the recommendations for the "Toy Story" movie are still the worst, with only one Toy Story recommendation. However, the other recommendations for the Toy Story movie are at least (more or less) the same genre.

Table 4: Predictions for Star Wars: Episode II - Attack of the Clones (2002).

Rank	**Title**	**Rating**	**Genres**
1	**Star Wars: Episode II - Attack of the Clones (2002)**	5.3498	Action,Sci-Fi
2	**Star Wars: Episode III - Revenge of the Sith (2005)**	4.9842	Action,Sci-Fi
3	Naruto Shippuden the Movie: Road to Ninja (2012)	4.9184	Action,Adventure,Fantasy
4	**Star Wars: Episode I - The Phantom Menace (1999)**	4.8043	Action,Sci-Fi
5	2012: Ice Age (2011)	4.4698	Action,Adventure,Sci-Fi
6	Dane Cook: Vicious Circle (2006)	4.0818	Comedy
7	Skins (2017)	4.0142	Comedy,Drama
8	**Star Wars: The Clone Wars (2008)**	3.7469	Action,Adventure,Animation,Sci-Fi
9	Terminal (2004)	3.5699	Drama,Thriller
10	Kevin Smith: Sold Out - A Threevening with Kevin Smith (2008)	3.5274	Comedy

Table 5: Predictions for Lord of the Rings: The Return of the King (2003).

Rank	**Title**	**Rating**	**Genres**
1	**Lord of the Rings: The Return of the King, The (2003)**	4.5797	Action,Adventure
2	**Lord of the Rings: The Two Towers, The (2002)**	4.5460	Adventure,Fantasy
3	**Lord of the Rings: The Fellowship of the Ring, The (2001)**	4.5226	Adventure
4	The Invisible Guardian (2017)	4.5072	Crime,Thriller
5	Mute (2018)	4.4020	Mystery,Sci-Fi,Thriller
6	Queen of Earth (2015)	4.0153	Drama
7	**Hobbit: An Unexpected Journey, The (2012)**	3.9652	Adventure,Fantasy,IMAX
8	Sailor Moon S the Movie: Hearts in Ice (1994)	3.9606	Action,Animation,Comedy
9	**The Hobbit: The Battle of the Five Armies (2014)**	3.9568	Adventure
10	**Hobbit: The Desolation of Smaug, The (2013)**	3.9563	Adventure

Table 6: Predictions for Toy Story (1995).

Rank	**Title**	**Rating**	**Genres**
1	Son of God (2014)	5.8015	Drama
2	Atharintiki Daaredi (2013)	5.7088	Action,Comedy,Drama
3	The Search (1948)	5.6446	Drama,War
4	The Putin Interviews (2017)	5.4298	(no genres listed)
5	The Fantastic Flying Books of Mr. Morris Lessmore (2011)	5.4061	Children
6	Oosaravelli (2011)	5.2345	Action,Romance
7	Patton Oswalt: Finest Hour (2011)	5.1999	Comedy
8	**Toy Story 2 (1999)**	5.1897	Adventure,Animation,Children
9	Neznaika na lune (1997)	5.1677	(no genres listed)
10	The Legend of Sleepy Hollow (1949)	5.0762	Animation,Comedy,Musical

6.4 The most and least polarizing movies

Note that the magnitude of an item trait vector has an effect on magnitude of the predicted rating for that movie. An item vector with a larger magnitude will in some cases lead to a larger inner product with certain user vectors.

Thus, the most polarizing movies are the ones which trait vectors have small magnitudes. Conversely, the least polarizing movies are the ones which trait vectors have small magnitudes. The magnitudes of the item trait vectors are calculated using the L1 norm (absolute sum) and the L2 norm (Euclidean norm).

6.4.1 Small dataset results

The following Tables (7 - 10) summarizes the most/least polarizing movies in the small dataset using the L1 and L2 norms .

Movies	**L1 vector norm**
Pay It Forward (2000)	18.5068693
How to Lose a Guy in 10 Days (2003)	18.40508315
Thin Red Line, The (1998)	17.8846111

Table 7: The three most polarizing movies, using L1 distance.

Movies	**L1 vector norm**
Christopher Columbus: The Discovery (1992)	1.45149368
Wind and the Lion, The (1975)	1.49269052
Fugitive, The (1947)	1.50684792

Table 8: The three least polarizing movies, using L1 distance.

Movies	**L2 vector norm**
Pay It Forward (2000)	5.29792418
Grifters, The (1990)	4.62169082
Hellboy (2004)	4.61594904

Table 9: The three most polarizing movies, using L2 distance.

Movies	**L2 vector norm**
Lilya 4-Ever (Lilja 4-ever) (2002)	0.42302444
Hellsinki (Rööperi) (2009)	0.42750923
Fugitive, The (1947)	0.43966882

Table 10: The three least polarizing movies, using L2 distance.

6.4.2 Full dataset results

The following Tables (11 - 14) summarizes the most/least polarizing movies in the full dataset using the L1 and L2 norms.

Movies	**L1 vector norm**
Birdemic: Shock and Terror (2010)	40.63732348
Room, The (2003)	38.70718889
Manos: The Hands of Fate (1966)	34.02765909

Table 11: The three most polarizing movies, using L1 distance.

Movies	**L1 vector norm**
64: Part 1 (2016)	1.67346213
64: Part 2 (2016)	1.68315035
The Lone Wolf Returns (1935)	1.68315035

Table 12: The three least polarizing movies, using L1 distance.

Movies	**L2 vector norm**
Room, The (2003)	11.00425097
Birdemic: Shock and Terror (2010)	10.95132207
Expelled: No Intelligence Allowed (2008)	9.21358835

Table 13: The three most polarizing movies, using L2 distance.

Movies	**L2 vector norm**
64: Part 1 (2016)	0.45170503
64: Part 2 (2016)	0.45850453
The Lone Wolf Returns (1935)	0.45850453

Table 14: The three least polarizing movies, using L2 distance.

One result that is particularly interesting to me is that “The Room” and “Birdemic: Shock and Terror” are the most polarizing movies in the full dataset (both for the L1 and L2 norm). The reason why I find this interesting is that both of these movies are seen by the Internet as bad movies that are “so bad that it’s good” (i.e. funny to watch). Adding to that, I think that if someone has rated on of these two movies it may say a lot about their personality.

7 Conclusion

This report presents a recommender system using Yehuda's matrix factorization method, which was implemented in `Python`.

There are several issues with this implementation. Collecting explicit feedback is challenging, since it is typically the case that the majority of users do not care for rating a movie. Collecting implicit feedback is a much more simple task, so for future work I would consider implementing a Bayesian personalized ranking [4] (BPR) system. The other issue with the implementation discussed in this report is that when inferring the trait vectors and the biases it was only a point estimate. The system could by improved by implementing a Variational Bayes [5] solution.

One big issue with my implementation is that I did not use a validation set schema. This meant that I could not implement cross-validation to properly tune the hyperparameters. The other issue is that there may be outliers in the data that can cause biased recommendations, and in future work potential outliers will be removed.

Overall, this implementation manages to handle large and sparse rating matrices. Yehuda's matrix factorization method is also relatively easy to implement and can be parallelized for efficient computation on large datasets.

References

[1] Ulrich Paquet is the course coordinator and he is referenced for his guidance during this project.

[2] Koren, Y. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.

[3] Harper, F. M., & Konstan, J. A. (2015). The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS), 5(4), Article 19. DOI: 10.1145/2827872.

[4] Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI'09), (pp. 452-461). AUAI Press.

[5] Koenigstein, N. and Paquet, U., 2013, October. Xbox movies recommendations: Variational Bayes matrix factorization with embedded feature selection. In Proceedings of the 7th ACM Conference on Recommender Systems (pp. 129-136).