AMERICAN UNIVERSITY OF BEIRUT

MAROUN SEMAAN FACULTY OF ENGINEERING & ARCHITECTURE

Department of Industrial Engineering & Management

ENMG 616 - Advanced Optimization Techniques & Algorithms
Final Project - Fair Movie Recommendation System

Written By:

Ralph Mouawad - ID 202204667

Jade Chabbouh - ID 202202353

To:

Dr. Maher Nouiehed

Copies of our MATLAB codes are presented in the Appendix

Contents

1 Introduction & Problem Formulation

Recommendation systems are algorithms aimed at predicting users' interests to recommend relevant products (such as movies, music, etc.). Their significance reverberates across diverse tech industries and online retail platforms, where their efficacy translates into substantial revenue streams. According to McKinsey, 35% of Amazon's revenue is generated through its recommendation system [1]. Additionally, according to most sources, the driving force behind Spotify's massive user base and lead in the music streaming industry isn't just the extensive music library it possesses, but the ingenious utilization of recommendations that weave a tailored musical experience for each individual [2]. In this paper, we aim to build a Movie Recommendation Engine to predict and recommend relevant movies to different users.

A collaborative filtering technique will be used, where predictions are made by looking at user-item relations: When identifying two similar users, the items preferred by the first user are recommended to the second user and vice versa. Concluding, fairness of the model will be defined, evaluated, and improved between males and females recommendations.

We are first given a list of m users $\{u_1, u_2, ..., u_m\}$ and n items $\{i_1, i_2, ..., i_n\}$, and the ratings of users towards items are represented by an incomplete $m \times n$ matrix $\mathbf{M}$, where $\mathbf{M}_{i,j}$ either represents the rating of user i for movie j, or is a missing value. We define $\Omega = \{(i,j) \| \mathbf{M}_{i,j} \text{ observed}\}$, the subset containing non-missing entries. We aim to recover a low-rank matrix using this subset. The optimization problem is as follows:

$$\min_{\mathbf{X} \in \mathbb{R}^{m \times n}} \text{rank}(\mathbf{X}) \text{ s.t. } \mathbf{X}_{i,j} = \mathbf{M}_{i,j} \text{ for } (i,j) \in \Omega.$$

[1] https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers

[2] https://neemz.medium.com/the-inner-workings-of-spotifys-ai-powered-music-recommendations-how-spotify-shapes-your-playlist-a10a9148ee8d

The rank of the matrix is the number of non-zero eigenvalues. Hence, we need to recover a matrix with the lowest possible rank that matches the observed rankings in Ω. This problem is extremely hard to solve, thus substituting the rank (number of non-zero eigenvalues) with the nuclear norm yields the following convex optimization problem

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times n}} \|\mathbf{X}\|_* \quad \text{s.t. } \mathbf{X}_{i,j} = \mathbf{M}_{i,j} \text{ for all } (i,j) \in \Omega,$$

which is more tractable. To allow for some flexibility in the constraint, a more common formulation of the problem is as follows

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times n}} \|\mathbf{X}\|_* + \lambda \sum_{(i,j)\in\Omega} \left(\mathbf{X}_{i,j} - \mathbf{M}_{i,j}\right)^2.$$

Despite convexity, our formulated problem can have an extremely high dimension $(m \times n)$. Moreover, estimating the nuclear norm requires computing the singular value decomposition at every iteration which is also computationally expensive.

Thus, we use the Matrix Factorization method, which enforces a low-rank matrix by expressing $\mathbf{X}$ as a product of two matrices $\mathbf{U} \in \mathbb{R}^{m\times r}$ and $\mathbf{V} \in \mathbb{R}^{n\times r}$:

$$\mathbf{X} = \mathbf{U}\mathbf{V}^{\mathbf{T}}.$$

This forces the rank of $\mathbf{X}$ to be at most r and yields the following optimization problem

$$\min_{\mathbf{U}\in\mathbb{R}^{m\times r},\mathbf{V}\in\mathbb{R}^{n\times r}} \sum_{(i,j)\in\Omega} \left(\mathbf{u}_i\mathbf{v}_j^\top - \mathbf{M}_{i,j}\right)^2$$

where $\mathbf{u}_i$ is the i^{th} row of $\mathbf{U}$ and $\mathbf{v}_j$ corresponds to the j^{th} row of $\mathbf{V}$. Furthermore, for

more stable training, bias terms are introduced which yields the following assumption

$$\mathbf{X}_{ij} = \mathbf{u}_i \mathbf{v}_j^\top + p_i + q_j,$$

where p_i and q_j are scalar bias terms for user i and movie j, respectively. This problem aims at recovering a matrix with rank at most r having similar observed entries. To prevent overfitting a regularization term is introduced to the squared error which yields

$$\min_{\mathbf{U}\in\mathbb{R}^{m\times r},\mathbf{V}\in\mathbb{R}^{n\times r}} \sum_{(i,j)\in\Omega} \left[\left(\mathbf{u}_i \mathbf{v}_j^\top + p_i + q_j - \mathbf{M}_{i,j} \right)^2 + \lambda \left(\|\mathbf{u}_i\|^2 + \|\mathbf{v}_j\|^2 \right) \right].$$

Note that the number of variables in the problem above is $(m+n)r$ which is much less than the dimension of the initial problem when $r << m, n$. In our problem, each row $\mathbf{u}_i$ represents a vector of features for user i and each row $\mathbf{v}_i$ represents a vector of features for item i. Hence $\mathbf{u}_i \mathbf{v}_j^\top + p_i + q_j$ is our prediction of the rating of user i to item j. The sections below explore

2 Implementation of the Solution

2.1 Data Pre-Processing

We first import the data contained in the file "MovieLens Dataset.csv". We then save the 'Ratings' sheet inside the matrix "Ratings" on MATLAB, then we proceed to shuffle the rows, making sure that individual ratings for specific movies remained unchanged.
See Appendix Section 5.1 for the MATLAB code.

2.2 Optimization

We randomly initialize $\mathbf{U} \in \mathbb{R}^{943\times r}, \mathbf{V} \in \mathbb{R}^{1682\times r}, \mathbf{p} \in \mathbb{R}^{943}$, and $\mathbf{q} \in \mathbb{R}^{1682}$ using a uniform distribution over $[0, 1]$. Here, r stands for the maximum feasible rank or number of features that will be extracted.

We then solve the optimization problem obtained using Stochastic Gradient Descent (SGD), by choosing an index 'i' corresponding to a data point at each iteration, i.e. one row of data in the file. We randomly select 'i', belonging to the range {1, size(data)}, then we compute the user ID, movie ID and the rating at this iteration. The following updates are performed at each iteration:

$$\begin{aligned}
\mathbf{u}_i &= \mathbf{u}_i - \alpha \left[2\left(\mathbf{u}_i\mathbf{v}_j^T + p_i + q_j - \mathbf{M}_{i,j}\right)\mathbf{v}_j + 2\lambda\mathbf{u}_i\right] \\
\mathbf{v}_j &= \mathbf{v}_j - \alpha \left[2\left(\mathbf{u}_i\mathbf{v}_j^T + p_i + q_j - \mathbf{M}_{i,j}\right)\mathbf{u}_i + 2\lambda\mathbf{v}_j\right], \\
p_i &= p_i - \alpha \left[2\left(\mathbf{u}_i\mathbf{v}_j^T + p_i + q_j - \mathbf{M}_{i,j}\right)\right] \\
q_j &= q_j - \alpha \left[2\left(\mathbf{u}_i\mathbf{v}_j^T + p_i + q_j - \mathbf{M}_{i,j}\right)\right],
\end{aligned}$$

where (i, j) are the User ID and Movie ID in the selected row of the matrix.

2.3 Evaluation of the Model

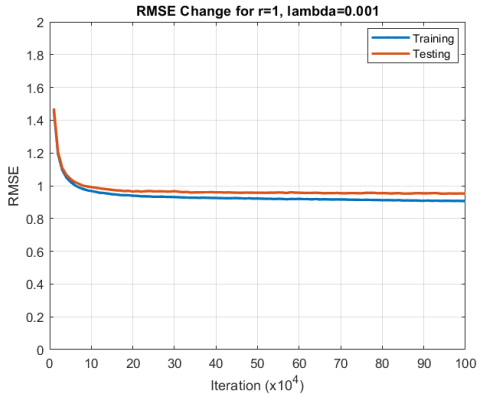
To evaluate our rating predictions, we use the Root Mean Squared Error (RMSE) which is defined as:

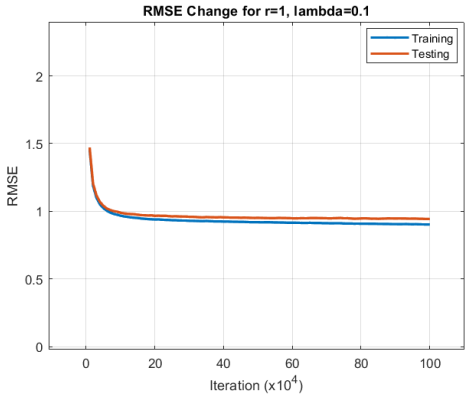
$$RMSE = \sqrt{\frac{1}{n}\sum_{i,j}\left(\mathbf{u}_i\mathbf{v}_j^\top + p_i + q_j - \mathbf{M}_{i,j}\right)^2},$$

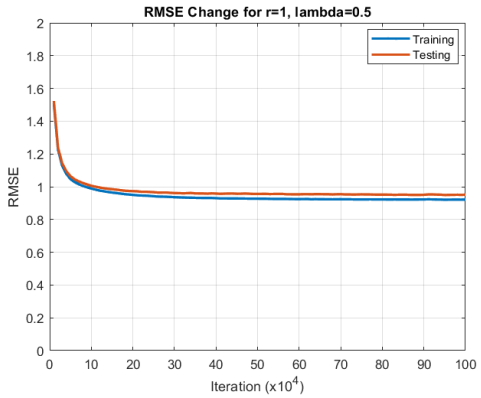
where n is the number of data points used for evaluation, and (i, j) are the (User Id, Movie id) in the corresponding n rows of data.

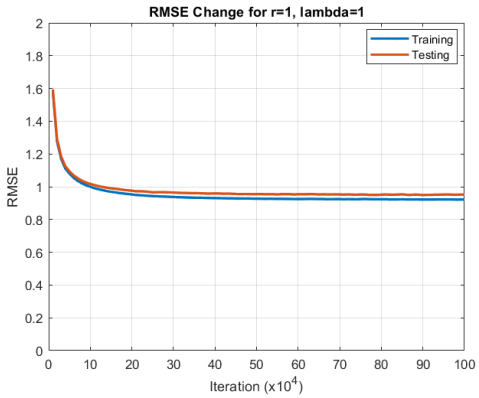
We then proceed by plotting the RMSE error for the training data and testing data, computed once every 10,000 iterations, while also using cross-validation to hyper-tune the value of λ. Below are the plots of training and testing data RMSE errors for $r = 1, 3, 5, 7, 9$, for different values of λ.

1. For $\mathbf{r = 1}$ and $\lambda = 0.001, 0.1, 0.5, 1$:

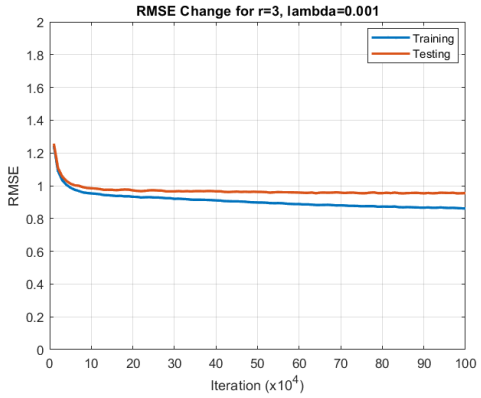


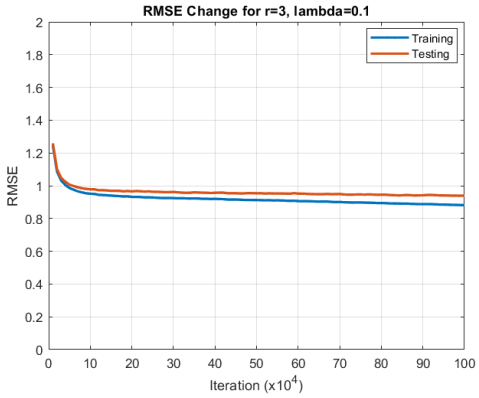


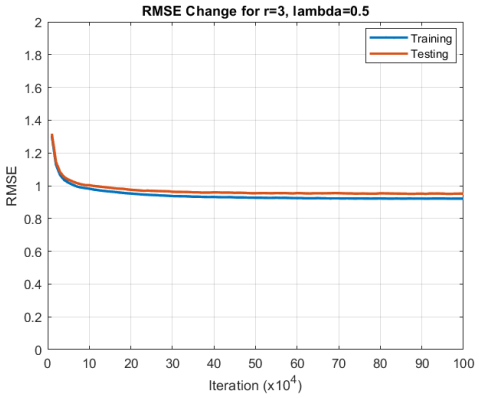


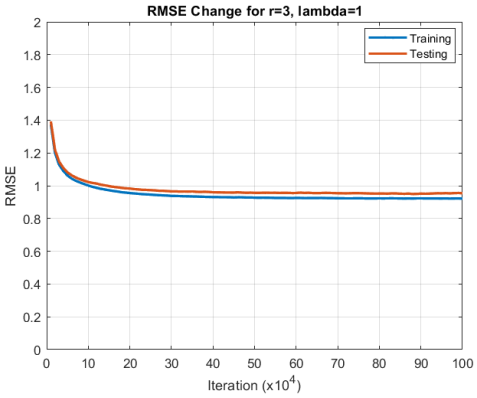


2. For $\mathbf{r = 3}$ and $\lambda = 0.001, 0.1, 0.5, 1$:

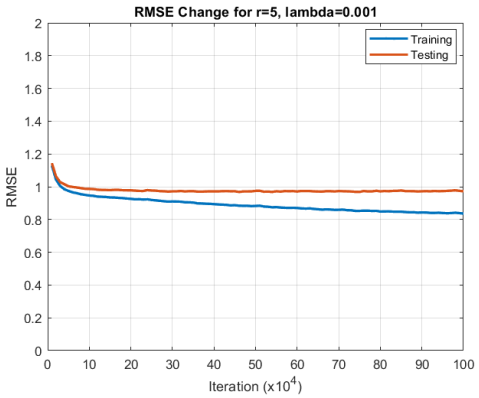


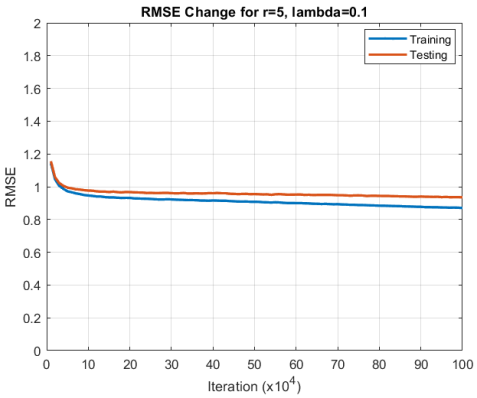


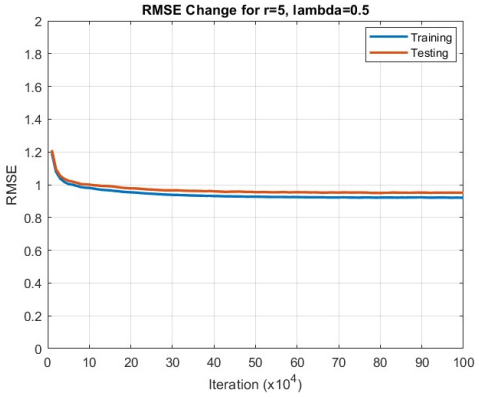


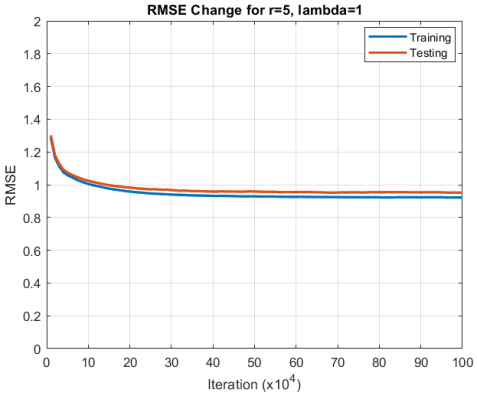


3. For $\mathbf{r = 5}$ and $\lambda = 0.001, 0.1, 0.5, 1$:

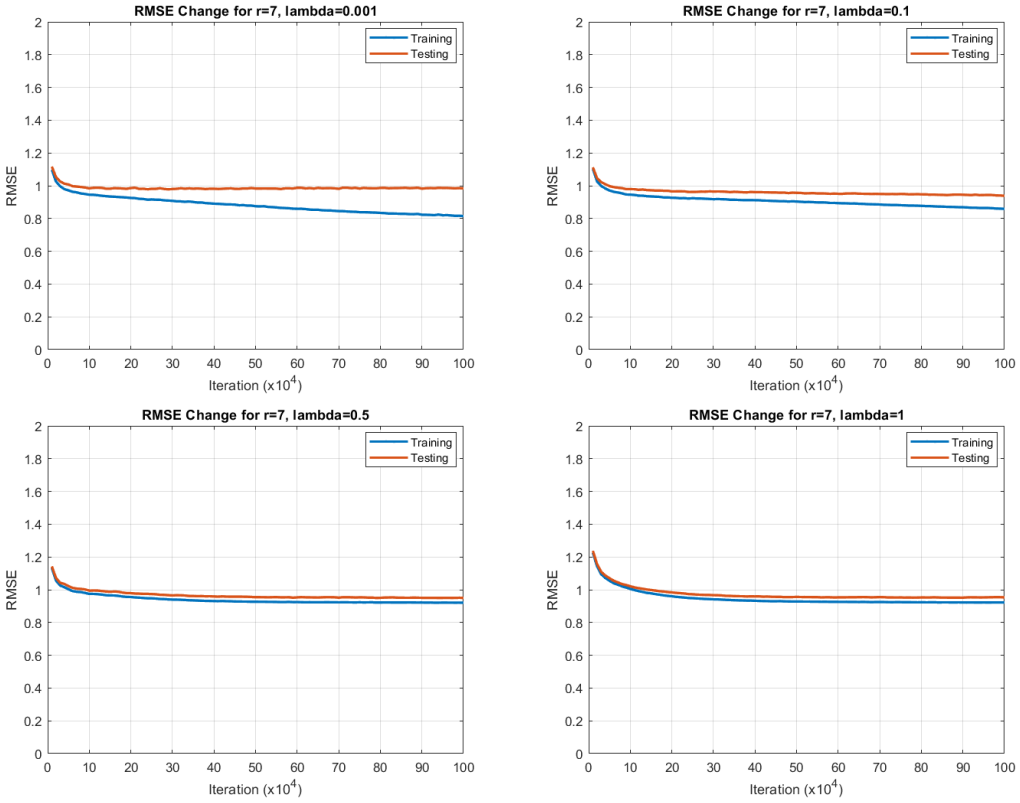




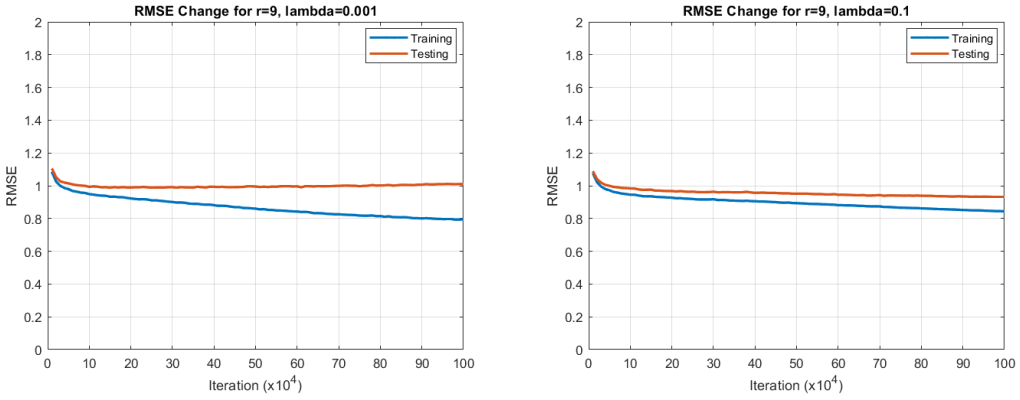


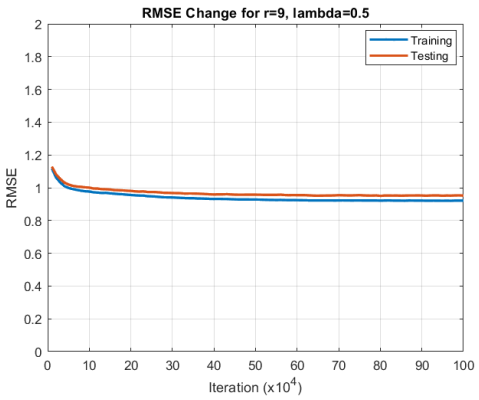


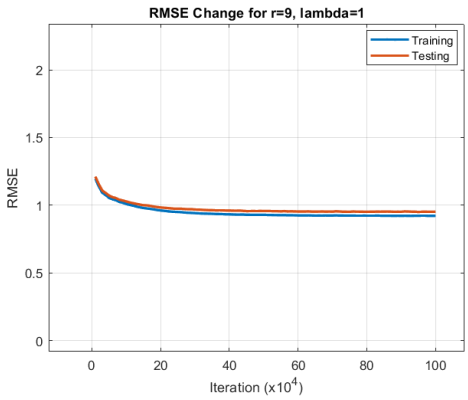
4. For $\mathbf{r = 7}$ and $\lambda = 0.001, 0.1, 0.5, 1$:



5. For $\mathbf{r = 9}$ and $\lambda = 0.001, 0.1, 0.5, 1$:







For each model, when we increase the regularization term λ, the difference between the training error and the testing error decreases because of the penalty imposed on complex models to prevent overfitting. The best regularization term to use here is $\lambda = 1$ since both training and testing errors decrease to the same value of approximately 0.8.

We can also notice that each time we increase the number of latent features r, we have an increased error on the testing data, especially when the regularization term is small. For example, moving from $\lambda = 0.001$ and r = 1 to $\lambda = 0.001$ and r = 9, the difference between the training and testing error increases from around 0.1 to 0.3. The increase in testing error with small λ with the increase of latent features is explained by the fact that our model is getting more complex to be able to catch better relations and do better predictions, but at the expense of an increased error. Increasing the regularization term, i.e, for $\lambda = 1$ and r = 9 will decrease the difference in error between training and testing data to around 0. The model that will be used for upcoming analysis is the one with a number of latent features $r = 9$, and a regularization term $\lambda = 1$ as it performed best.

Also note that the algorithm always converged for all values of r and λ.

2.4 Feature Extraction

Each row of the matrix $\mathbf{V}$ corresponds to a movie and each column corresponds to a feature. Our goal is understanding what these features correspond to, and we extracted the movies with the highest value of each feature. This was done by selecting movies having at least 25 ratings for a more accurate analysis. We fixed r = 9 throughout. Below are the steps followed:

Features

First, we define the feature index we want to analyze in matrix $\mathbf{V}$, as well as the number of top movies we want to extract and the minimum ratings each film should have to be considered. We chose to extract 10 top movies, and we want a minimum rating of 25 per film.

Then, we sort the feature column we are analyzing in the matrix $\mathbf{V}$ in descending order, and we store the sorted indices in a list.

We initiate a loop over the sorted indices, that checks if each of these sorted movies has at least 25 ratings and adds it to a list named 'topMovies' until we have 10 top movies.
For each movie in the loop, we retrieve the movie ID and check if it had at least 25 ratings by counting how many times this movie ID occurred in the 'Ratings' matrix data. If the above checks, i.e. the movie has at least 25 ratings, we store it.

Genre

To get the genre of each of out top 10 movies, we accessed each movie stored in our 'topMovies' list. We went into its corresponding row in the sheet 'Movies' and iterated over the different columns (the genres). If at the movie row, a column entry is '1', it means that this genre is attributed to the specific film. We will store column number (-) 2, to get the

name of the genre from the Genre sheet. After getting the list, we iterate over the genre list for each movie, and replace the genre's ID by their names.

Below are the tables displaying the name of each of the top 10 movies extracted for each feature with its associated genre.

Movie Name	**Genre ID**	**Genre Name**
Anna Karenina (1997)	9 – 15	Fantasy, Sci-Fi
Super Mario Bros. (1993)'	2 – 3 – 5 – 16	Adventure, Animation, Comedy, Thriller
Spice World (1997)	6 – 13	Crime, Mystery
Keys to Tulsa (1997)	7	Documentary
Homeward Bound II: Lost in San Francisco (1996)	3 – 5	Animation, Comedy
FairyTale: A True Story (1997)	5 – 9 – 10	Comedy, Fantasy, Film-Noir
Air Bud (1997)	5 - 6	Comedy, Crime
House Arrest (1996)	6	Crime
Pillow Book, The (1995)	9 - 15	Fantasy, Sci-Fi
Aladdin and the King of Thieves (1996)	4 - 5 - 6	Children's, Comedy, Crime

Table 1: Feature Extraction for column 4 of $\mathbf{V}$

Movie Name	Genre ID	Genre Name
Anna Karenina (1997)	9 – 15	Fantasy, Sci-Fi
Exotica (1994)	9	Fantasy
Girl 6 (1996)	6	Crime
Ghost in the Shell (Kokaku kidotai) (1995)	4 – 16	Children's, Thriller
Super Mario Bros. (1993)	2 – 3 – 5 – 16	Adventure, Animation, Comedy, Thriller
Blues Brothers 2000 (1998)	2 – 6 – 13	Adventure, Crime, Mystery
That Darn Cat! (1997)	5 – 6 – 14	Comedy, Crime, Romance
Living in Oblivion (1995	6	Crime
Deceiver (1997)	7	Documentary
Spice World (1997)	6 - 13	Crime, Mystery

Table 2: Feature Extraction for column 5 of $\mathbf{V}$

Movie Name	Genre ID	Genre Name
Blues Brothers 2000 (1998)	2 – 6 – 13	Adventure, Crime, Mystery
Anna Karenina (1997)	9 – 15	Fantasy, Sci-Fi
Deceiver (1997)	7	Documentary
Ghost in the Shell (Kokaku kidotai) (1995)	4 – 16	Children's, Thriller
That Darn Cat! (1997)	5 – 6 – 14	Comedy, Crime, Romance
Exotica (1994)'	9	Fantasy
Portrait of a Lady, The (1996)	9	Fantasy
Spice World (1997)	6 – 13	Crime, Mystery
Washington Square (1997)	9	Fantasy
FairyTale: A True Story (1997)	5 – 9 - 10	Comedy, Fantasy, Film-Noir

Table 3: Feature Extraction for column 8 of $\mathbf{V}$

Interpretation of Results

Each feature column from matrix **V** displays a small portion of genres. In fact, each column displays one or two genres as a majority, and some other features as a minority. Since some movies are considered to have multiple genres, they are displayed in more than one feature column. Below is the analysis for each column feature:

- **Feature column 4:** The most recurring genre is 5 – 'Comedy'. We have 5 films that are considered to have some comedy in it. Genre 6 – 'Crime' is also contained in 4 out of the 10 films we have. This means that this latent feature takes into consideration 'Comedy' and 'Crime' mostly. Other genres are also displayed but at a lower level, such as 7, 10 or 13. Genre 9 – 'Fantasy' is displayed three times, making it a medium contributor.

- **Feature column 5:** The most recurring genre is number 6 – 'Crime', displayed 5 times. This means that this feature column takes 'Crime' as one of the most important genres. We also have other genres, as a minority, such as genre 16 – 'Thriller' displayed twice, as well as genres 9, 2 and 13. These are minor contributors in this feature column.

- **Feature column 8:** The most recurring genre is number 9 – 'Fantasy', displayed 5 times. This means that this feature column of matrix **V** is taking 'Fantasy' as a main genre. We also have other minor genres displayed, such as 5, 13, 4, 16 etc. Note that genre 6 – "Crime" – is displayed 3 times, making it a medium contributor.

We can also notice that some movies appear more than once, because they belong to many different genres, such as "Anna Karenina", belonging to genres 9 & 15. These 2 genres appear on the 3 columns that we analyzed, meaning that they always impact all feature columns analyzed.

2.5 Recommendation of Similar Movies

Besides finding personalized recommendations for users, a common task of movie recommendation systems is recommending similar items. For instance, YouTube recommends a list of similar videos for each video. To create such a list, we measure the similarity between two movies using the following expression

$$\text{Similarity }_{i,j} = \frac{\langle \mathbf{v}^i, \mathbf{v}_j \rangle}{\|\mathbf{v}_i\| \, \|\mathbf{v}_j\|}.$$

This measures the similarity of movies i and j by measuring the cosine of the angle between the features of these two movies. To compute the similarity list of a movie, we fixed $r = 9$ and selected an arbitrary movie with at least 25 ratings. Below are the steps followed:

We first re-generate a movie matrix $\mathbf{V}$ as some rows were sorted in the previous section. Then, we selected an arbitrary movie 'i' with at least 25 ratings to find its similar list in the matrix $\mathbf{V}$. We create a list to store all the movies of the matrix with the cosine similarity score obtained by measuring it with the selected movie.

We iterate over all movies of the matrix $\mathbf{V}$: at each row (or movie) 'j', we calculated the cosine between the features of movie 'j' and the features of movie 'i'. We then stored the ID 'j', with the similarity score obtained with movie 'i', in the list. We also made sure that indices 'i' and 'j' are different to not measure the similarity between the same movie as it will obviously be the most similar.

At the end of the loop, we obtain a list with all the movies in the first column, and the cosine measure for each movie in the second column.

We iterate over the list obtained in descending order based on the second column, i.e.

the scores, to obtain the similarities scores in descending order. We then create a Top 5 similar movies for our selected movie 'i'. We also replace the IDs by the names of the movies using the 'Movies' matrix containing the movies IDs, names, and features.

The arbitrarily selected movie was ID 500 for our test run. The results obtained are displayed in the below table. We also compute the genres of each movie to see whether obtained similar movies belong to similar genres.

Movie ID	Movie Name	Genres ID	Genres Names
500	**Fly Away Home (1996)**	**2, 4**	**Adventure, Children's**
1009	Stealing Beauty (1996)	8	Drama
1032	Little Big League (1994)	4, 5	Children, Comedy
1099	Red Firecracker, Green Firecracker (1994)	8	Drama
898	Postman, The (1997)	8	Drama
773	Mute Witness (1994)	17	War

Table 4: Top 5 similar movies to movie ID 500

Interpretation of Results

The arbitrarily selected movie was 'Fly Away Home', combining both 'Adventure' and 'Children's' genres. One of the similar movies recommended features 'Children's' and 'Comedy' as genres. This might indicate that viewers who enjoy 'Children's' movies may also appreciate films that combine 'Children's' and 'Comedy' elements.

An essential observation is that most of the recommended similar movies belong to 'Drama' genre (Genre 8). A reason for that might be that when we generate matrices $\mathbf{U}$ and $\mathbf{V}$ for users and movies, we look at the users' ratings for different movies of different genres. The fact that recommending similar movies to one that involves 'Adventure' and 'Children's' includes 'Drama' movies can indicate that users who watch 'Adventure' and 'Children's' movies also tend to watch or enjoy 'Drama' movies. This could be a pattern captured re-

lated to the combination of movies watched by users, i.e. there's a strong pattern between 'Adventure' and 'Drama', as 'Drama' appears to be a preference among users who tend to watch 'Adventure' movies.

Another recommended similar movie is 'Mute Witness', which belongs to 'War' genre (Genre 17). This could be another pattern of a combination of movies watched by users: 'Adventure', 'Drama', and 'War'.

While there may be various combinations to explore (such as 'Children's' and 'War', 'Drama' and 'War', etc.), the ones discussed above seem particularly relevant in this case.

3 Fair Movie Recommendation Engine

In this section, we will introduce the notion of fairness in machine learning. It is the process of eliminating biases in a trained model, based on gender or race. In our case, we will measure the fairness of our trained recommendation system in predicting the ratings of movies for different genders (males and females) for the following four genres: *Romance, Action, Sci-Fi* and *Musical*.

Let $\mathcal{U}_m$ and $\mathcal{U}_f$ be the set of male and female users, respectively. Moreover, let $\mathcal{G}_g$ be the set of movies with genre $g \in \{$*Romance, Action, Sci-Fi, Musical*$\}$. Then we define the fairness metric per genre as follows

$$\mathcal{F}_g(\mathbf{U}, \mathbf{V}, \mathbf{p}, \mathbf{q}) = \left(\frac{1}{|\mathcal{U}_m|\,|\mathcal{G}_g|} \sum_{i \in \mathcal{U}_m} \Big(\sum_{j \in \mathcal{G}_g} \Big(\mathbf{u}_i \mathbf{v}_j^\top + p_i + q_j \Big) \Big) \left(\frac{1}{|\mathcal{U}_f|\,|\mathcal{G}_g|} \sum_{i \in \mathcal{U}_f} \Big(\sum_{j \in \mathcal{G}_g} \mathbf{u}_i \mathbf{v}_j^\top + p_i + q_j \right) \Big) \right)^2$$

for $g \in \{$*Romance, Action, Sci-Fi, Musical*$\}$.

3.1 Measure of Fairness

To measure the fairness of our model, we compute the difference in the average predicted rating of male and female users for movies belonging to the previously selected genres — *Romance, Action, Sci-Fi, and Musical*.

We sort users into two matrices, $\mathbf{U}_{males}$ and $\mathbf{U}_{females}$, where each row represents a male or female user, and each column denotes a latent feature. Each of these two matrices has 9 columns as we use $r = 9$ throughout. Bias terms for males and females, denoted as p_{males} and $p_{females}$ are also computed. To do that, we iterate over the 'Users' sheet in our dataset: For each user, we add the corresponding row of the $\mathbf{U}$ matrix (computed at the beginning) displaying all users with their latent features, to the matrix $\mathbf{U}_{males}$ if the user is a male, or to the matrix $\mathbf{U}_{females}$ otherwise.

Next, we compute matrices for each of the selected genres and their latent features. Iterating over the 'Movies' sheet, we check the entry for each movie in columns dedicated to the specified genres. If the entry is '1', indicating the movie belongs to the genre, we store the corresponding entry from the initial matrix $\mathbf{V}$, containing the latent features of each movie. Additionally, bias terms q for each genre of movies were computed.

Below are the steps to compute the average rating of each genre for both genders. This procedure was done for each of the selected genres:

Let's assume we want to get the average predicted rating of males for *Romance* movies:

- We first iterate over all male users - for $i = 1$ to the size of $\mathbf{U}_{\text{males}}$
- Then, we iterate for each user over all the movies in the *Romance* matrix - for $j = 1$ to the size of $\mathbf{G}_{\text{romance}}$
- At each movie iteration, we compute the below term and add it to a cumulative

rating number:

$$\Big(U_{\text{males}(i,:)}G^{\top}_{\text{romance}(j,:)} + p(i) + q(j)\Big)$$

- At the end, we divide by the multiplication of the sizes of $\mathbf{U}_{males}$ and $G_{romance}$, resulting in the computation of the following term:

$$\frac{1}{\|U_{\text{males}}\|\|G_{\text{romance}}\|}\sum_{i\in U_{\text{males}}}\sum_{j\in G_{\text{romance}}}\Big(U_{\text{males}(i,:)}G^{\top}_{\text{romance}(j,:)} + p(i) + q(j)\Big)$$

This term represents the average predicted ratings of males to *Romance* movies. We perform the same steps for both males and females and for all of the four selected genres. The below table summarizes the obtained results:

	Romance	**Action**	**Sci-Fi**	**Musical**
Average Rating Male	3.2593	3.0084	3.1478	3.339
Average Rating Female	3.3035	3.0515	3.1902	3.39

Table 5: Male vs Female rating prediction for selected genres

	Romance	**Action**	**Sci-Fi**	**Musical**
Average Rating Male	3.67	3.4	3.67	3.5
Average Rating Female	3.65	3.4	3.52	3.66

Table 6: Male vs Female actual rating for selected genres

As we can see, the predicted average rating of females is always slightly higher than the predicted average rating of males. This issue would raise concerns about the fairness of our model in predicting ratings for both genders. In our case, it is not a very high difference, but in some cases, one should take care of biases in machine learning algorithms. We will try to impose fairness on our trained model by adding a regularization term that penalizes the difference in average predicted ratings.Regarding the actual average ratings, we can see

first that our predictions are a little bit different, and this mainly due to the fact that we took 9 latent features. Maybe if we choose to get more features, we could have predicted more accurate ratings. However, our prediction are not very bad. We can see in the actual ratings, that sometimes, males give higher ratings than females, sometimes it's the opposite or females giving a higher rating. In our predictions, it's always the females that rate higher $even if slightly$. This is why we will try to implement fairness on our model.

3.2 Imposing Fairness

The fairness metric that will be added to the optimization problem to penalize these differences is:

$$\mathcal{F}_g(\mathbf{U},\mathbf{V},\mathbf{p},\mathbf{q}) = \left(\frac{1}{|\mathcal{U}_m|\,|\mathcal{G}_g|}\sum_{i\in\mathcal{U}_m}(\sum_{j\in\mathcal{G}_g}\left(\mathbf{u}_i\mathbf{v}_j^\top + p_i + q_j\right)) - \left(\frac{1}{|\mathcal{U}_f|\,|\mathcal{G}_g|}\sum_{i\in\mathcal{U}_f}(\sum_{j\in\mathcal{G}_g}\mathbf{u}_i\mathbf{v}_j^\top + p_i + q_j\right))\right)^2$$

We re-initialize the matrices **U, V, p, q** and, store in separate vectors the male users ID, female users ID, romance movies ID, action movies ID, sci-fi movies ID and musical movies ID. When implementing the Stochastic Gradient Descent algorithm, we will check whether the selected user is a male or female, as well as the genre of the movie selected.
We repeat the steps of data pre-processing by shuffling the rows and splitting the data into training and testing.

3.2.1 Optimization

In this part, we update the optimization problem by adding a regularization term that penalizes the differences between average predicted ratings. The optimization problem becomes:

$$\min_{U\in\mathbb{R}^{m\times r},V\in\mathbb{R}^{n\times r}}\sum_{(i,j)\in\Omega}(\mathbf{u_i}\mathbf{v_j}^\top + \mathbf{p_i} + \mathbf{q_j} - \mathbf{M_{i,j}})^2 + \lambda(\|u_i\|_2^2 + \|v_j\|_2^2) + \lambda_2\sum_g \mathbf{F_g}(\mathbf{U},\mathbf{V},\mathbf{p},\mathbf{q})$$

To solve the above problem, we randomly initialize $\mathbf{U} \in \mathbb{R}^{943\times r}, \mathbf{V} \in \mathbb{R}^{1682\times r}, \mathbf{p} \in \mathbb{R}^{943}$, and $\mathbf{q} \in \mathbb{R}^{1682}$ using a uniform distribution over $[0, 1]$. Here r stands for the maximum feasible rank or number of features that will be extracted.

We then use Stochastic Gradient Descent (SGD), choosing a random index corresponding to a data point, first checking if the movie belongs to one of the four selected genres, and if it does, checking whether the user is male or female. If the user is a male for example, we randomly choose a female with her predicted rating for that movie, and we compute the gradient of the fairness term $\nabla\mathbf{F_g}$, and vice-versa. If the movie doesn't fall under one of the four selected genres, we compute the regular SGD update rules, disregarding the fairness term $\mathbf{F}_g$.

The gradient of the fairness metric is of the form $\nabla F_g(u, v, p, q) = (\frac{\partial F_g}{\partial u_i}, \frac{\partial F_g}{\partial v_j}, \frac{\partial F_g}{\partial p_i}, \frac{\partial F_g}{\partial q_j})$. The fairness term is of the form a^2, thus its derivative would be of the form $2aa'$. Below are the gradient computations with respect to u, v, p, and q

$$\frac{\partial F_g}{\partial u_i} = 2\left[\frac{1}{|G_g|}\sum_{j\in G_g}(u_i v_j^T + p_i + q_j) - \frac{1}{|G_g|}\sum_{j\in G_g}(u_{i'} v_j^T + p_{i'} + q_j)\right]\left(\frac{1}{G_g}\sum_{j\in G_g}(v_j^T)\right),$$

$$\frac{\partial F_g}{\partial v_j} = 2\left[\frac{1}{U_m}\sum_{i\in U_m}(u_i v_j^T + p_i + q_j) - \frac{1}{U_f}\sum_{i\in U_f}(u_{i'} v_j^T + p_{i'} + q_j)\right]\left(\frac{1}{|U_m|}\sum_{i\in U_m}(u_i) - \frac{1}{|U_f|}\sum_{i\in U_f}(u_i)\right),$$

$$\frac{\partial F_g}{\partial p_i} = 2\left[\frac{1}{G_g}\sum_{j\in G_g}(u_i v_j^T + p_i + q_j) - \frac{1}{G_g}\sum_{j\in G_g}(u_{i'} v_j^T + p_{i'} + q_j)\right]\frac{|G_g|}{|G_g|},$$

$$\frac{\partial F_g}{\partial q_j} = 2\left[\frac{1}{U_m}\sum_{i\in U_m}(u_i v_j^T + p_i + q_j) - \frac{1}{U_f}\sum_{i\in U_f}(u_i v_j^T + p_i + q_j)\right](\frac{|U_m|}{|U_m|} - \frac{|U_f|}{|U_f|}) = 0.$$

3.2.2 Simplification for SGD

To apply stochastic gradient descent, the batch size is reduced to 1, corresponding to a single user and movie pair. This simplification removes the summations over users and movies, and the equations are further simplified. The updated gradients are derived as follows:

Case 1: $i \in U_m \rightarrow i' \in U_f$

1. For $\frac{\partial F_g}{\partial u_i}$:

$$\frac{\partial F_g}{\partial u_i} = 2\left((u_i v_j^T + p_i + q_j) - (u_i' v_j^T + p_i' + q_j)\right) \cdot v_j^T$$

2. For $\frac{\partial F_g}{\partial v_j}$:

$$\frac{\partial F_g}{\partial v_j} = 2\left((u_i v_j^T + p_i + q_j) - (u_i' v_j^T + p_i' + q_j)\right) \cdot u_i$$

3. For $\frac{\partial F_g}{\partial p_i}$:

$$\frac{\partial F_g}{\partial p_i} = 2\left((u_i v_j^T + p_i + q_j) - (u_i' v_j^T + p_i' + q_j)\right) \cdot 1$$

4. For $\frac{\partial F_g}{\partial q_j}$ (same for all cases):

$$\frac{\partial F_g}{\partial q_j} = 2\,(\text{—-})\,(1-1) = 0$$

Case 2: $i \in U_f \rightarrow i' \in U_m$

1. For $\frac{\partial F_g}{\partial u_i}$:

$$\frac{\partial F_g}{\partial u_i} = 2\left((u_i v_j^T + p_i + q_j) - (u_i v_j^T + p_i' + q_j)\right) \cdot v_j^T$$

2. For $\frac{\partial F_g}{\partial v_j}$:

$$\frac{\partial F_g}{\partial v_j} = 2\left((u_i v_j^T + p_i + q_j) - (u_i v_j^T + p_i' + q_j)\right) \cdot u_i$$

3. For $\frac{\partial F_g}{\partial p_{i'}}$ (switch i and i'):

$$\frac{\partial F_g}{\partial p_{i'}} = 2\left((u_{i'} v_j^T + p_{i'} + q_j) - (u_i v_j^T + p_i + q_j)\right) \cdot 1$$

4. For $\frac{\partial F_g}{\partial q_j}$ (same for all cases):

$$\frac{\partial F_g}{\partial q_j} = 2\,(—)\,(1-1) = 0$$

These simplified gradients ensure efficient updates when applying stochastic gradient descent to enforce fairness during training. The next step involves integrating these gradients into the optimization algorithm and implementing the fairness-aware recommendation system.

In our code, we are computing the predicted ratings for one male and one female for the same movie. This means not all of these summations will be included inside the codes, i.e. one computation at a time.

3.2.3 Algorithm Adjustment

After deriving the fairness-aware gradients and defining the fairness metric, we moved on to updating the recommendation algorithm to incorporate gender-based fairness considerations. The adjustments ensure that the model minimizes prediction bias across genders, particularly for selected genres.

Identification of User and Genre Groups

To enable fairness-aware updates, we first identified specific user and movie groups required for the optimization process:

- **User Groups:** The set of male and female user IDs was extracted based on the gender attribute in the user dataset. This allowed us to distinguish between user groups and implement fairness-aware comparisons.
- **Genre-based Movie Groups:** For the selected genres (*Romance*, *Action*, *Sci-Fi*, and *Musical*), the corresponding movie IDs were identified by checking the genre labels in the movie dataset.

This preprocessing step created lists of user and movie IDs that were later used to calculate fairness adjustments during optimization.

Fairness-aware Stochastic Gradient Descent (SGD)

We extended the stochastic gradient descent algorithm to include fairness constraints. The adjustments involved the following:

- **Opposite Gender Comparison:** During each iteration, for a randomly selected user, an opposite-gender user was also sampled. This allowed us to introduce a fairness-aware comparison.
- **Genre-specific Adjustment:** For movies belonging to the target genres, fairness constraints were applied. Specifically, the gradient updates for the user feature matrix U and the movie feature matrix V were modified to include fairness regularization terms.
- **Parameter Updates:** The gradients were adjusted with regularization terms weighted by the fairness hyperparameter λ_2. The learning rate was progressively decayed to ensure convergence.

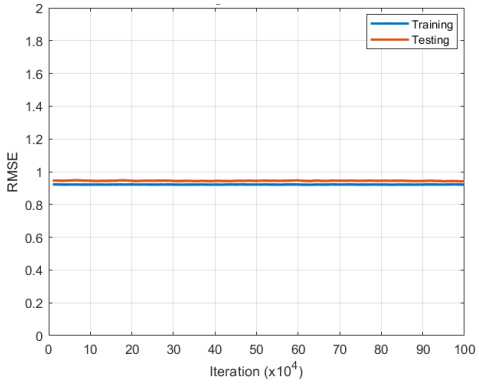
The fairness-aware updates ensure that the disparity in average predicted ratings across genders is minimized while maintaining accurate predictions.

3.2.4 Evaluation

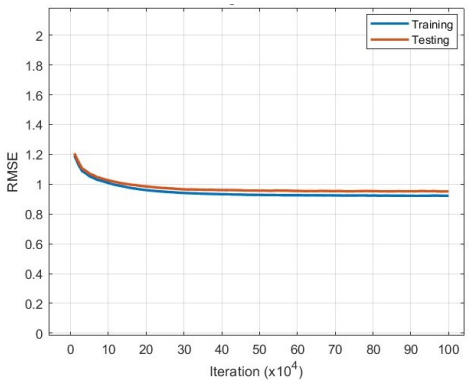
We evaluate our new predictions using the Root Mean Squared Error (RMSE) used in previous sections (Check Section 2.3 for the full error definition).

We then proceed by plotting the RMSE error for the training data and testing data, computed once every 10,000 iterations, while also using cross-validation to hyper-tune the value of λ_2.
Below are the plots of training and testing data RMSE errors for $r = 9$, for different values of λ_2.

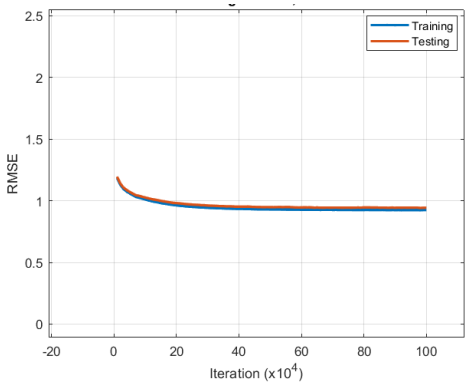
1. $\lambda_2 = 0.01$:



2. $\lambda_2 = 0.5$:



3. $\lambda_2 = 1$:



We can see that when we increase the regularization term, the results obtained are better. There's no difference between training error and testing error. For $\lambda_2 = 0.5$, the results were approximately as good as if it was 1. When we took $\lambda 2 = 0.01$, the plot was not very accurate. This might be due to an issue with the code as it turned out very complex.
We have tried to plot the change of the fairness term for the genre 'Romance'. However, due to the complexity of our code, we got corrupted values. We would expect, however, to see this value decrease to less than 0.05 $difference\,obtained\,before$ when the number of iterations increase.

3.2.5 Results

After implementing SGD and including the fairness term, we note the average gender-based ratings for each of the selected genres are slightly closer, with differences for genres not exceeding .002, meaning that the model has minimized the differences observed in Table 5 (Section 3.1). This results in a fairer model that penalizes differences in gender-based ratings for most of the four selected genres.

Below are the results obtained after adding the fairness term $\mathcal{F}_g$ and computing the average ratings using the new vales of **U, V, p,** and **q**.

	Romance	**Action**	**Sci-Fi**	**Musical**
Average Rating Male	3.3268	3.0265	3.243	3.436
Average Rating Female	3.3124	3.0115	3.334	3.447

Table 7: Male vs Female rating prediction for selected genres

For future work, we can try to talk about other notions of fairness, such as disparate mistreatment. This new type was introduced in previous papers where a penalty term would be for error rates between males and females, and not for the actual scores. Sometimes, sensitive attributes would play a role in ratings.

4 Appendix - MATLAB codes

4.1 Data Pre-Processing

```
Ratings = readcell('MovieLens Dataset.xlsx', 'Sheet','Ratings', 'Range','A2:C100001'); % access the sheet Ratings in the dataset.
Ratings(1:5,:,:); % display the first 5 rows

numRows = size(Ratings, 1); % size of the matrix (nb of rows)
permutedIndices = randperm(numRows); % permutation of indices

shuffled_Ratings = Ratings(permutedIndices, :); % shuffle the rows of the matrix

isDataUnchanged = isequal(sortrows(Ratings), sortrows(shuffled_Ratings)); % making sure the data didnt change
if isDataUnchanged
    disp('Data in each row remains the same after shuffling.'); % make sure the rating of user 'i' to movie 'j' is still the same
else
    disp('Data in each row has changed after shuffling.');
end

shuffled_Ratings(1:5,:,:); % the data didnt change and the rows are shuffled
Ratings(1:5,:,:);
Ratings = shuffled_Ratings; % Now the Ratings rows are shuffled and we can split our data into train/test.

% Split into Training and Testing
percentageTraining = 0.8;
numTraining = round(percentageTraining * numRows); %nb of rows of training data

Ratings_training = Ratings(1:numTraining, :); % training matrix/ dataset
Ratings_testing = Ratings(numTraining+1:end, :); % testing matrix/ dataset
```

4.2 Optimization and Evaluation

4.2.1 Optimization

```
Ratings_training; Ratings_testing; % our training & testing datasets
r = 9; % maximum number of features to be extracted
alpha = 0.01; % step size
lambda = 1; % regularization

% Initialize the matrices
U = rand(943,r);
V = rand(1682,r);
p = rand(943,1);
q = rand(1682,1);
RMSE_training = zeros(1,100);
counter = 0;
iteration = 10000;
RMSE_training_index =1;
RMSE_test = zeros(1,100);
RMSE_test_index = 1;
```

```
% Using the Stochastic Gradient Descent algorithm (choose 1 random index)
for i=1:1000000
    randomIndex = randi(size(Ratings_training, 1)); % pick random index from the Ratings dataset for SGD
    userID = Ratings_training(randomIndex,1); % get the index we're going to use for matrix U
    movieID = Ratings_training(randomIndex,2); % get the index we're going to use for matrix V
    rating = Ratings_training(randomIndex,3); % get the actual rating

    userID = cell2mat(userID); % the data extracted was in a cell format we converted them into numerical
    movieID = cell2mat(movieID);
    rating = cell2mat(rating);

    % Update rules for each
    error = U(userID,:)*V(movieID,:)' + p(userID) + q(movieID) - rating;
    U_update = 2*(error)*V(movieID,:) + 2*lambda*U(userID,:);
    V_update = 2*(error)*U(userID,:) + 2*lambda*V(movieID,:);
    p_update = 2*(error);
    q_update = 2*(error);

    % Updating each term
    U(userID,:) = U(userID,:) - alpha*U_update;
    V(movieID,:) = V(movieID,:) - alpha*V_update;
    p(userID) = p(userID) - alpha*p_update;
    q(movieID) = q(movieID) - alpha*q_update;

    counter = counter + 1;

    if counter == iteration
        iteration = iteration + 10000;
        RMSE_training(RMSE_training_index) = getError(U, V, p, q, Ratings_training); % get the training error
        RMSE_training_index = RMSE_training_index + 1; % update the index
        RMSE_test(RMSE_test_index) = getError(U, V, p, q, Ratings_testing); % testing error
        RMSE_test_index = RMSE_test_index +1;
    end
end
```

```
% Plotting RMSE change over iterations
figure;

% Plot RMSE (training)
plot(1:100, RMSE_training, 'LineWidth', 2, 'DisplayName', 'Training');
hold on;

% Plot RMSE (testing)
plot(1:100, RMSE_test, 'LineWidth', 2, 'DisplayName', 'Testing');

title(['RMSE Change for r=' num2str(r) ', lambda=' num2str(lambda)]);
xlabel('Iteration (x10^4)'); % Adjusted xlabel
ylabel('RMSE');
grid on;
legend('Location', 'Best');
ylim([0, 2]);

hold off;
```

4.2.2 RMSE function

```
function rmse = getError(U, V, p, q, Data) % we commented this code bcz

    n = size(Data,1);

    cum_error = 0;
    for i = 1:n
        user_ID = cell2mat(Data(i,1));
        movie_ID = cell2mat(Data(i,2));
        rating2 = cell2mat(Data(i,3));
        error2 = U(user_ID,:)*V(movie_ID,:)' + p(user_ID) + q(movie_ID) - rating2;
        cum_error = error2^2 + cum_error;
    end
    rmse = sqrt((1/n)*cum_error);
end
```

4.3 Feature Extraction

```
V; % We are using the one with r = 9 and lambda = 1 bcz it had the best performance

% Let's extract the sheet Movies from our dataset.
Movies = readcell('MovieLens Dataset.xlsx', 'Sheet','Movies', 'Range','A2:U1683'); % access the sheet and read it
Movies(1:5,1:3); % display some films to make sure the file is well extracted
Genre = readcell('MovieLens Dataset.xlsx', 'Sheet','Genres','Range','A2:B20');

featureIndex = 4; % Feature index to analyze
numTopMovies = 10; % Number of top movies to extract for each feature
minRatings = 25; % Minimum ratings for accurate analysis

% Sort movies based on Feature Index of the matrix V in descending order
[~, sortedMovieIndices] = sort(V(:, featureIndex), 'descend');

% Select top movies with at least 25 ratings
topMovies = [];
for movieIndex = sortedMovieIndices' % loop over the sorted indices
    movieID = Movies{movieIndex, 1}; % select the movie
    % Check if the movie has at least 25 ratings
    if sum(cell2mat(Ratings(:, 2)) == movieID) >= minRatings % if the film has at leat 25 ratings
        topMovies = [topMovies; Movies(movieIndex, 2)]; % Add the movie to the list
        if size(topMovies, 1) == numTopMovies
            break; % Stop when we have enough top movies
        end
    end
end

% Display the results
disp(['Top movies for Feature ' num2str(featureIndex) ':']);
disp(topMovies)

% Get the genre IDs of each movie

% Create a cell array to store genres for each top movie
topGenresList = cell(numTopMovies, 1);

% Iterate over the topMovies list
for i = 1:numTopMovies
    movieName = topMovies{i}; % Get the movie name from the topMovies list

    % Find the row index of the movie in the Movies matrix
    movieRowIndex = find(strcmp(Movies(:, 2), movieName));

    % Initialize a list to store genre IDs for the current movie
    currentGenres = [];

    % Check each genre column (column 3 to 21)
    for genreIndex = 3:21
        if cell2mat(Movies(movieRowIndex, genreIndex)) == 1
            % If the entry is 1, it means the genre is associated with the movie
            currentGenres = [currentGenres, genreIndex - 2]; % Store the genre ID
        end
    end

    % Store the genre IDs for the current movie in the topGenresList
    topGenresList{i} = currentGenres;
end

% Display the genre IDs for each top movie
disp('Genre IDs for Each Top Movie:');
disp(topGenresList); % we got the genre ID for each film (check report for their names).
```

4.4 Recommending Similar Movies

```
selectedFilmIndex = 500; % choose the movie we want to find similar ones to

% Check if the selected movie has at least 25 ratings, if not we have to
% select another one
if sum(cell2mat(Ratings(:, 2)) == selectedFilmIndex) < 25
    disp(['The selected movie (Index ' num2str(selectedFilmIndex) ') does not have at least 25 ratings.']);
    return; % Exit the code if the condition is not met
else
    disp(['The selected movie (Index ' num2str(selectedFilmIndex) ') has 25 or more ratings and is relevant for analysis.']);
end
% We've re-run the code to get a new vector V bcz some columns were sorted
V; % The movie matrix with latent features


selectedFilmIndex = 500; % choose the movie we want to find similar ones to
v_i = V(selectedFilmIndex, :); % take the features of this movie from the movie matrix V

%Initiate a list of similar movies for 'i'
similarMoviesList = [];


for j = 1:size(V, 1) % Loop over the matrix V
    if j ~= selectedFilmIndex % make sure we dont compare the same movies since they'll be very similar with cos =1
        % Calculate the cosine similarity between vi and vj
        similarity_ij = dot(v_i, V(j, :)) / (norm(v_i) * norm(V(j, :)));
        similarMoviesList = [similarMoviesList; j, similarity_ij]; % add the movie j with tis simil value to the list (now has 2 col)
    end
end

%Sort the list by descending order of similarity values
similarMoviesList = sortrows(similarMoviesList, -2);

% Make the list called 'topSimilarMovies' by selecting the first 5 movie IDs
topSimilarMovies = similarMoviesList(1:5, 1);

% Display the results
disp(['Top 5 Similar Movies to Selected Movie (Index ' num2str(selectedFilmIndex) '):']);
disp(topSimilarMovies);

% Access the names of the top similar movies
topSimilarMovieNames = Movies(topSimilarMovies, 2);

% Access the name of the selected movie
selectedMovieName = Movies(selectedFilmIndex, 2);

% Display the results
disp(['Selected Movie Name: ' selectedMovieName]);
disp('Top 5 Similar Movies:');
disp(topSimilarMovieNames);

% check the interpretation of the results in the report.
```

4.5 Measure of Fairness

4.5.1 Predicted Ratings

2- Measuring average predictions for males

```
g = G_musical; m2 = size(g,1); q2 = q_musical; % change these if we want to see the avg prediction for another genre
% Avergae Rating Preditions for males -
m1 = size(U_males,1); cum_rating_males = 0;
for i=1:m1
    for j = 1:m2
        R_males = U_males(i,:)*g(j,:)' + p_males(i) + q2(j);
        cum_rating_males = cum_rating_males + R_males;
    end
end
average_rating_male = cum_rating_males/(m1*m2);
disp('Average Predicted Rating for Males');
disp(average_rating_male);
```

3- Measuring average predictions for females

```
g = G_musical; m2 = size(g,1); q2 = q_musical;
n1 = size(U_females,1); cum_rating_females = 0;
for i=1:n1
    for j = 1:m2
        R_females = U_females(i,:)*g(j,:)' + p_females(i) + q2(j);
        cum_rating_females = cum_rating_females + R_females;
    end
end
average_rating_female = cum_rating_females/(n1*m2);
disp('Average Predicted Rating for Females');
disp(average_rating_female);
```

4.5.2 Actual Ratings

Actual ratings of males and females

```
avgMaleAction = 0;
avgMaleRomance = 0;
avgMaleScifi = 0;
avgMaleMusical = 0;

avgFemaleAction = 0;
avgFemaleRomance = 0;
avgFemaleScifi = 0;
avgFemaleMusical = 0;

totalMaleAction = 0;
totalMaleRomance = 0;
totalMaleScifi = 0;
totalMaleMusical = 0;

totalFemaleAction = 0;
totalFemaleRomance = 0;
totalFemaleScifi = 0;
totalFemaleMusical = 0;

for i = 1:length(Ratings)
    if ismember(Ratings{i, 1}, malesID)
        if ismember(Ratings{i, 2}, romanceID)
            avgMaleRomance = avgMaleRomance + Ratings{i, 3};
            totalMaleRomance = totalMaleRomance + 1;
        elseif ismember(Ratings{i, 2}, actionID)
            avgMaleAction = avgMaleAction + Ratings{i, 3};
```

```
            totalMaleAction = totalMaleAction + 1;
        elseif ismember(Ratings{i, 2}, scifiID)
            avgMaleScifi = avgMaleScifi + Ratings{i, 3};
            totalMaleScifi = totalMaleScifi + 1;
        elseif ismember(Ratings{i, 2}, musicalID)
            avgMaleMusical = avgMaleMusical + Ratings{i, 3};
            totalMaleMusical = totalMaleMusical + 1;
        end
    elseif ismember(Ratings{i, 1}, femalesID)
        if ismember(Ratings{i, 2}, romanceID)
            avgFemaleRomance = avgFemaleRomance + Ratings{i, 3};
            totalFemaleRomance = totalFemaleRomance + 1;
        elseif ismember(Ratings{i, 2}, actionID)
            avgFemaleAction = avgFemaleAction + Ratings{i, 3};
            totalFemaleAction = totalFemaleAction + 1;
        elseif ismember(Ratings{i, 2}, scifiID)
            avgFemaleScifi = avgFemaleScifi + Ratings{i, 3};
            totalFemaleScifi = totalFemaleScifi + 1;
        elseif ismember(Ratings{i, 2}, musicalID)
            avgFemaleMusical = avgFemaleMusical + Ratings{i, 3};
            totalFemaleMusical = totalFemaleMusical + 1;
        end
    end
end

avgMaleMusical/totalMaleMusical
```

4.6 Imposing Fairness

Data Pre-Processing (same as before)

```
Ratings = readcell('MovieLens Dataset.xlsx', 'Sheet','Ratings', 'Range','A2:C100001'); % access the sheet Ratings in the dataset.
Ratings(1:5,:,:); % display the first 5 rows

numRows = size(Ratings, 1); % size of the matrix (nb of rows)
permutedIndices = randperm(numRows); % permutation of indices

shuffled_Ratings = Ratings(permutedIndices, :); % shuffle the rows of the matrix

isDataUnchanged = isequal(sortrows(Ratings), sortrows(shuffled_Ratings)); % making sure the data didnt change
if isDataUnchanged
    disp('Data in each row remains the same after shuffling.'); % make sure the rating of user 'i' to movie 'j' is still the same
else
    disp('Data in each row has changed after shuffling.');
end

shuffled_Ratings(1:5,:,:); % the data didnt change and the rows are shuffled
Ratings(1:5,:,:);
Ratings = shuffled_Ratings; % Now the Ratings rows are shuffled and we can split our data into train/test.

% Split into Training and Testing
percentageTraining = 0.8;
numTraining = round(percentageTraining * numRows); %nb of rows of training data

Ratings_training = Ratings(1:numTraining, :); % training matrix/ dataset
Ratings_testing = Ratings(numTraining+1:end, :); % testing matrix/ dataset

G_musical(1:5,:)
```

Find the IDs for each gender and for each genre of movies

```
U = rand(943,r); V = rand(1682,r); p = rand(943,1); q = rand(1682,1); r =9;
Users = readcell('MovieLens Dataset.xlsx', 'Sheet', 'Users', 'Range', 'A2:D944'); % read the Users Sheet
malesID = []; femalesID =[]; romanceID =[]; scifiID = []; actionID = []; musicalID =[];

for i = 1:size(Users,1)
    if strcmp(Users(i, 3), 'M') % if it's a male
        malesID = [malesID; cell2mat(Users(i,1))];
    end
    if strcmp(Users(i, 3), 'F') % if it's a female
        femalesID = [femalesID; cell2mat(Users(i,1))];
    end
end

for i = 1:size(Movies,1)
    if cell2mat(Movies(i,17)) == 1 %romance
        romanceID = [romanceID; cell2mat(Movies(i,1))];
    end
    if cell2mat(Movies(i, 4)) == 1 % action
        actionID = [actionID; cell2mat(Movies(i,1))];
    end
    if cell2mat(Movies(i,18)) ==1 % sci-fi
        scifiID = [scifiID; cell2mat(Movies(i,1))];


    end
    if cell2mat(Movies(i, 15)) %musical
        musicalID = [musicalID; cell2mat(Movies(i,1))];
    end
end
size(malesID)
```

```
%% Optimization with new penalty term
alpha = 0.01;
lambda1 = 1;
lambda2 = 1.01; % for the second regularization term
RMSE_training = zeros(1,100);
counter = 0;
counter2 = 0; %to plot diff in fairness each 100k iterations
iteration = 10000;
iteration2 = 100000;
FairnessList = zeros(1,10);
RMSE_training_index =1;
RMSE_test = zeros(1,100);
RMSE_test_index = 1;
for i=1:1000000
    male_rating = 0; female_rating = 0; cum_male_rating =0; cum_female_rating = 0;
    randomIndex = randi(size(Ratings_training, 1)); % pick random index from the Ratings dataset for SGD
    userID = cell2mat(Ratings_training(randomIndex,1)); % get the index we're going to use for matrix U
    movieID = cell2mat(Ratings_training(randomIndex,2)); % get the index we're going to use for matrix V
    rating = cell2mat(Ratings_training(randomIndex,3)); % get the actual rating

    error = U(userID,:)*V(movieID,:)' + p(userID) + q(movieID) - rating;
    % Update rules for each
    if ismember(movieID, romanceID)
        if ismember(userID, malesID(1)) % if it's a male, randomly select a female and impose fairness for this specific movie
            male_rating = U(userID, :) * V(movieID, :)' + p(userID) + q(movieID);
            femID = length((femalesID));
            female_rating = U(femalesID(femID), :) * V(movieID, :)' + p(femalesID(femID)) + q(movieID);
        elseif ismember(userID, femalesID)
            female_rating = U(userID, :) * V(movieID, :)' + p(userID) + q(movieID);
            maID = length((malesID)); %random ID for male
            male_rating = U(malesID(maID), :) * V(movieID, :)' + p(malesID(maID)) + q(movieID);
        end
        fairness_term_romance = 2 * (male_rating - female_rating);
        U_update = 2 * (error) * V(movieID, :) + 2 * lambda1 * U(userID, :);
        if ismember(userID, malesID(1)) %we do this to know what male ID we use in the update rule of V
            V_update = 2 * (error) * U(userID, :) + 2 * lambda1 * V(movieID, :) + 2 * lambda2 * fairness_term_romance * (U(userID, :) - U(femalesID(femID),:));
        else
            V_update = 2 * (error) * U(userID, :) + 2 * lambda1 * V(movieID, :) + 2 * lambda2 * fairness_term_romance * (U(malesID(maID), :) - U(userID,:));
        end
        p_update = 2 * (error) ;
        q_update = 2 * (error) ;
    end


    if ismember(movieID, musicalID)% we're doing this to compute the fairness metric when we obtain a movie of one of the specific genres
        if ismember(userID, malesID(1))
            male_rating = U(userID, :) * V(movieID, :)' + p(userID) + q(movieID);
            femID = length((femalesID));
            female_rating = U(femalesID(femID), :) * V(movieID, :)' + p(femalesID(femID)) + q(movieID);
        elseif ismember(userID, femalesID(1))
            female_rating = U(userID, :) * V(movieID, :)' + p(userID) + q(movieID);
            maID = length((malesID));
            male_rating = U(malesID(maID), :) * V(movieID, :)' + p(malesID(maID)) + q(movieID);
        end
        fairness_term_musical = 2 * (male_rating - female_rating);
        U_update = 2 * (error) * V(movieID, :) + 2 * lambda1 * U(userID, :);
```

```
    U_update = 2 * (error) * V(movieID, :) + 2 * lambda1 * U(userID, :);
    if ismember(userID, malesID(1))
        V_update = 2 * (error) * U(userID, :) + 2 * lambda1 * V(movieID, :) + 2 * lambda2 * fairness_term_musical * (U(userID, :) - U(femalesID(femID),:));
    else
        V_update = 2 * (error) * U(userID, :) + 2 * lambda1 * V(movieID, :) + 2 * lambda2 * fairness_term_musical * (U(malesID(maID), :) - U(userID,:));
    end
    p_update = 2 * (error) ;
    q_update = 2 * (error) ;
end


if ismember(movieID, scifiID)% we're doing this to compute the fairness metric when we obtain a movie of one of the specific genres
    if ismember(userID, malesID(1))
        male_rating = U(userID, :) * V(movieID, :)' + p(userID) + q(movieID);
        femID = length((femalesID));
        female_rating = U(femalesID(femID), :) * V(movieID, :)' + p(femalesID(femID)) + q(movieID);
    elseif ismember(userID, femalesID(1))
        female_rating = U(userID, :) * V(movieID, :)' + p(userID) + q(movieID);
        maID = length((malesID));
        male_rating = U(malesID(maID), :) * V(movieID, :)' + p(malesID(maID)) + q(movieID);
    end
    fairness_term_scifi = 2 * (male_rating - female_rating);
    U_update = 2 * (error) * V(movieID, :) + 2 * lambda1 * U(userID, :);
    if ismember(userID, malesID(1))
        V_update = 2 * (error) * U(userID, :) + 2 * lambda1 * V(movieID, :) + 2 * lambda2 * fairness_term_scifi * (U(userID, :)-U(femalesID(femID),:));
    else
        V_update = 2 * (error) * U(userID, :) + 2 * lambda1 * V(movieID, :) + 2 * lambda2 * fairness_term_scifi * (U(malesID(maID), :) - U(userID,:));
    end
    p_update = 2 * (error) ;
    q_update = 2 * (error) ;
end

if ismember(movieID, actionID)% we're doing this to compute the fairness metric when we obtain a movie of one of the specific genres
    if ismember(userID, malesID(1))
        male_rating = U(userID, :) * V(movieID, :)' + p(userID) + q(movieID);
        femID = length((femalesID));
        female_rating = U(femalesID(femID), :) * V(movieID, :)' + p(femalesID(femID)) + q(movieID);
    elseif ismember(userID, femalesID(1))
        female_rating = U(userID, :) * V(movieID, :)' + p(userID) + q(movieID);
        maID = length((malesID));
        male_rating = U(malesID(maID), :) * V(movieID, :)' + p(malesID(maID)) + q(movieID);
    end
    fairness_term_action = 2 * (male_rating - female_rating);
    U_update = 2 * (error) * V(movieID, :) + 2 * lambda1 * U(userID, :);
    if ismember(userID, malesID)
        V_update = 2 * (error) * U(userID, :) + 2 * lambda1 * V(movieID, :) + 2 * lambda2 * fairness_term_action * (U(userID, :)-U(femalesID(femID),:));
    else
        V_update = 2 * (error) * U(userID, :) + 2 * lambda1 * V(movieID, :) + 2 * lambda2 * fairness_term_action * (U(malesID(maID), :) - U(femalesID(femID),:));
    end
    p_update = 2 * (error) ;
    q_update = 2 * (error) ;
else
    U_update = 2*(error)*V(movieID,:) + 2*lambda1*U(userID,:);
    V_update = 2*(error)*U(userID,:) + 2*lambda1*V(movieID,:);
    p_update = 2*(error);
```

```
    p_update = 2*(error);
    q_update = 2*(error);
end
% Updating each term
U(userID,:) = U(userID,:) - alpha*U_update;
V(movieID,:) = V(movieID,:) - alpha*V_update;
p(userID) = p(userID) - alpha*p_update;
q(movieID) = q(movieID) - alpha*q_update;
counter = counter + 1;
counter2 = counter2 +1;
if counter == iteration
    iteration = iteration + 10000;
    RMSE_training(RMSE_training_index) = getError(U, V, p, q, Ratings_training); % get the training error
    RMSE_training_index = RMSE_training_index + 1; % update the index
    RMSE_test(RMSE_test_index) = getError(U, V, p, q, Ratings_testing); % testing error
    RMSE_test_index = RMSE_test_index +1;
end
if counter2 == iteration2
    iteration2 = iteration2 + 100000;
    for i = 1:size(Movies, 1)
        G_musical = []; q_musical = [];
        if cell2mat(Movies(i, 15)) == 1 % check if the film is musical
            G_musical = [G_musical; V(i,:)]; %get latent features of musical movies
            q_musical = [q_musical; q(i)];
        end
        % latent features of males and females at this nb of iterations
        U_males =[]; % initialize the vectors
        U_females =[];
        p_males = [];
        p_females = [];
        for i = 1:size(Users, 1) % iterate over the Users matrix
            if strcmp(Users{i, 3}, 'M') % if it's a male, add the the latent features of this male to the vector of males
                U_males = [U_males; U(i, :)]; % each row will now have the latent features of one male user
                p_males = [p_males; p(i)];
            end
            if strcmp(Users{i, 3}, 'F') % same procedure for females
                U_females = [U_females; U(i,:)];
                p_females = [p_females; p(i)];
            end
        end
        %avg prediction males
        g = G_musical; m2 = size(g,1); q2 = q_musical; % change these if we want to see the avg prediction for another genre

        m1 = size(U_males,1); cum_rating_males = 0;
        for i=1:m1
            for j = 1:m2
                R_males = U_males(i,:)*g(j,:)' + p_males(i) + q2(j);
                cum_rating_males = cum_rating_males + R_males;
            end
        end
        average_rating_male = cum_rating_males/(m1*m2);
        %avg rating females
        g = G_musical; m2 = size(g,1); q2 = q_musical;
        n1 = size(U_females,1); cum_rating_females = 0;
```

```
                if strcmp(users{i}, 'F') % same procedure for females
                    U_females = [U_females; U(i,:)];
                    p_females = [p_females; p(i)];
                end
            end
            %avg prediction males
            g = G_musical; m2 = size(g,1); q2 = q_musical; % change these if we want to see the avg prediction for another genre

            m1 = size(U_males,1); cum_rating_males = 0;
            for i=1:m1
                for j = 1:m2
                    R_males = U_males(i,:)*g(j,:)' + p_males(i) + q2(j);
                    cum_rating_males = cum_rating_males + R_males;
                end
            end
            average_rating_male = cum_rating_males/(m1*m2);
            %avg rating females
            g = G_musical; m2 = size(g,1); q2 = q_musical;
            n1 = size(U_females,1); cum_rating_females = 0;
            for i=1:n1
                for j = 1:m2
                    R_females = U_females(i,:)*g(j,:)' + p_females(i) + q2(j);
                    cum_rating_females = cum_rating_females + R_females;
                end
            end
            average_rating_female = cum_rating_females/(n1*m2);

            f = (average_rating_male - average_rating_female)^2;
            FairnessList(i) = f;
        end
    end

end

% Plotting RMSE change over iterations
figure;

% Plot RMSE (training)
plot(1:100, RMSE_training, 'LineWidth', 2, 'DisplayName', 'Training');
hold on;

% Plot RMSE (testing)
plot(1:100, RMSE_test, 'LineWidth', 2, 'DisplayName', 'Testing');

title(['RMSE Change for r=' num2str(r) ', lambda=' num2str(lambda)]);
xlabel('Iteration (x10^4)'); % Adjusted xlabel
ylabel('RMSE');
grid on;
legend('Location', 'Best');
ylim([0, 2]);

hold off;

plot(1:10, FairnessList, 'LineWidth',2, 'DisplayName','Change of fairness value w.r.t epochs')

function rmse = getError(U, V, p, q, Data) % we commented this code bcz
                        %it was doing an error for the code after
    n = size(Data,1);         % if u want to run it, comment everything
                              % below and run
    cum_error = 0;
    for i = 1:n
        user_ID = cell2mat(Data(i,1));
        movie_ID = cell2mat(Data(i,2));
        rating2 = cell2mat(Data(i,3));
        error2 = U(user_ID,:)*V(movie_ID,:)' + p(user_ID) + q(movie_ID) - rating2;
        cum_error = error2^2 + cum_error;
    end
    rmse = sqrt((1/n)*cum_error);
end
```

```
Users = readcell('MovieLens Dataset.xlsx', 'Sheet', 'Users', 'Range', 'A2:D944'); % read the Users Sheet
U_males =[]; % initialize the vectors
U_females =[];
p_males = [];
p_females = [];
for i = 1:size(Users, 1) % iterate over the Users matrix
    if strcmp(Users{i, 3}, 'M') % if it's a male, add the the latent features of this male to the vector of males
        U_males = [U_males; U(i, :)]; % each row will now have the latent features of one male user
        p_males = [p_males; p(i)];
    end
    if strcmp(Users{i, 3}, 'F') % same procedure for females
        U_females = [U_females; U(i,:)];
        p_females = [p_females; p(i)];
    end
end
U_males(1:5,:); % display some results to see if it's correct
U_females(1:5,:); % it's correct because we have 9 columns (of the latent features) and Umales & Ufemales together have 943 users
% Display the results
disp('Male Users:');
disp(U_males);
disp(p_males);
disp(p_females)

disp('Female Users:');
disp(U_females);


% Make the vectors containing the latent features of  the movies of each
% genre (same procedure like users but we look here at Movies matrix)
G_romance = [] ; G_action = []; G_scifi = []; G_musical = [];
q_romance = []; q_action = []; q_scifi = []; q_musical = [];
% Romance is col 17, Action is col 4, Scifi is col 18 and Musical is col 15
%
for i = 1:size(Movies, 1)
    if cell2mat(Movies(i, 17)) == 1 %check if the film is romance
        G_romance = [G_romance; V(i, :)];
        q_romance = [q_romance; q(i)];
    end
    if cell2mat(Movies(i, 4)) == 1 % check if the film is action
        G_action = [G_action; V(i, :)]; % add the latent features to the corresponding vector
        q_action = [q_action; q(i)];
    end
    if cell2mat(Movies(i, 18)) == 1 % check if the film is sci-fi
        G_scifi = [G_scifi; V(i,:)];
        q_scifi = [q_scifi; q(i)];
    end
    if cell2mat(Movies(i, 15)) == 1 % check if the film is musical
        G_musical = [G_musical; V(i,:)];
        q_musical = [q_musical; q(i)];
    end
end % Now we have vectors containing males, females and the movies of each different genre with their latent features
% Since here we dont really need the IDs, we didnt include the ID of each
% user but we could have made a column containing the IDs
```

2- Measuring average predictions for males

```
g = G_scifi; m2 = size(g,1); q2 = q_scifi; % change these if we want to see the avg prediction for another genre
% Avergae Rating Preditions for males -
m1 = size(U_males,1); cum_rating_males = 0;
for i=1:m1
    for j = 1:m2
        R_males = U_males(i,:)*g(j,:)' + p_males(i) + q2(j);
        cum_rating_males = cum_rating_males + R_males;
    end
end
average_rating_male = cum_rating_males/(m1*m2);
disp('Average Predicted Rating for Males');
disp(average_rating_male);
```

3- Measuring average predictions for females

```
g = G_scifi; m2 = size(g,1); q2 = q_scifi;
n1 = size(U_females,1); cum_rating_females = 0;
for i=1:n1
    for j = 1:m2
        R_females = U_females(i,:)*g(j,:)' + p_females(i) + q2(j);
        cum_rating_females = cum_rating_females + R_females;
    end
end
average_rating_female = cum_rating_females/(n1*m2);
disp('Average Predicted Rating for Females');
disp(average_rating_female);
```