Movie Recommender – Documentation

Team Members and Work Distribution

- Project Proposal & Progress Report: Minjun Gao, Yixiang Cao
- Frontend Interface: Minjun Gao
- Recommendation Algorithm: Yixiang Cao
- Documentation: Minjun Gao, Yixiang Cao
- Video Presentation: Minjun Gao, Yixiang Cao

Overview

The project is a movie recommendation system that uses collaborative filtering to generate recommendations for users. The system is based on the theme of free choice, and it is designed to recommend movies to each user recorded in our chosen dataset. The dataset consists of movie ratings from 610 different users. When a user accesses our website, they can input their user number, specify the genres of movies they are interested in, and choose the number of recommended movies they would like to receive. The system will then generate a list of up to 20 movie recommendations, ranked in order of predicted favorability. The favorability values are predicted by training the Root Mean Square Error (RMSE) algorithm on the dataset, which calculates the distance between the true and predicted values for each movie.

Implementation

Frontend Interface (index.php)

The website, which is generated by index.php, enables users to enter three types of input: their **username**, which is their user number in the range from 1 to 610; the **genre** of the movies they are interested in, such as comedy, drama, romance, horror, sci-fi, etc., which should be capitalized; and the **number of recommended movies** they want to receive, which must be in the range from 1 to 20.

After a user inputs their desired group of values, such as 10, Comedy, 5, they will receive a result similar to the example in the following picture. This result will provide five recommended comedy movies for user number 10.

Search Result

Title	Genre
The True Memoirs of an International Assassin (2016)	Action|Comedy
Jim Jefferies: I Swear to God (2009)	Comedy
You'll Never Get Rich (1941)	Comedy|Musical|Romance
Letter to Three Wives, A (1949)	Comedy|Drama
Trip, The (2002)	Comedy|Drama|Romance

Recommender (`recommender_system.ipynb`)

The project uses collaborative filtering, implemented in the Python programming language within Jupyter Notebook, to generate movie recommendations. The process of creating the recommender system can be divided into the following steps:

I. To begin, the project imports a CSV dataset of movie ratings from 610 different users and converts it into a matrix with 9724 rows (representing the movies) and 610 columns (representing the users). Since not every user has rated every movie, most of the values in the matrix are represented as "N/A". We calculated the sparsity, or the percentage of "N/A" values, of the matrix and found it to be 98.30%. We then replaced all of the "N/A" values with zeros.

```
In [366]: ## change the format of ratings from DataFrame to matrix
          matrix_ratings = df_ratings.pivot_table(index = ['movieId'],columns = ['userId'],values = ['rating'])

In [367]: matrix_ratings
```

Out[367]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	
movieId																		
1	0.777778	NaN	NaN	NaN	0.777778	NaN	0.888889	NaN	NaN	NaN	...	0.777778	NaN	0.777778	0.555556	0.777778	0.444444	0.777
2	NaN	NaN	NaN	NaN	NaN	0.777778	NaN	0.777778	NaN	NaN	...	NaN	0.777778	NaN	1.000000	0.666667	NaN	N
3	0.777778	NaN	NaN	NaN	NaN	1.000000	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
4	NaN	NaN	NaN	NaN	NaN	0.555556	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
5	NaN	NaN	NaN	NaN	NaN	1.000000	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	0.555556	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
193581	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
193583	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
193585	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
193587	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
193609	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N

9724 rows × 610 columns

II. In this step, we assume that each movie and each user has 10 features. We initialize the movie and user parameters using the numpy.random.randint function, and we get the indexes of the non-zero values in the matrix. We also define a function named "rmse" to calculate the Root Mean Square Error (RMSE) using the formula provided below.

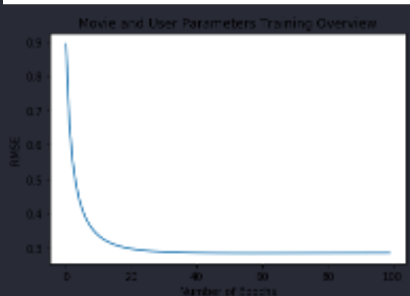
```
In [372]: ## get the index of non-zero values
          non_zero_movies,non_zero_users = matrix_ratings.values.nonzero()
```

```
In [373]: ## define rmse function
          ## rmse is to measure the distance between true value and predict value
          def rmse(predict, true):
              predict = predict[true.values.nonzero()]
              true = true.values[true.values.nonzero()]
              return np.sqrt(np.mean((predict - true) ** 2))
```

$$RMSE = \sqrt{(f - o)^2}$$

III. In this step, we use the Gradient Descent Algorithm to update the parameters and minimize the RMSE values. This part of the code updates the predictions 100 times, and it keeps track of the RMSE values in a list called "rmse_ls". We can plot a graph of the changing RMSE values as the number of epochs increases to visualize the training process, and we find that the RMSE values are indeed decreasing over time.

```
In [374]: %%time
          ## create rmse list, which stort rmse for each epoches
          rmse_ls = []
          ## update paramsters for 100 times
          for i in range(100):
              #print(i)
              for movie_idx, user_idx in zip(non_zero_movies,non_zero_users):
                  real_rating = matrix_ratings.values[movie_idx,user_idx]
                  predict_rating = np.dot(movie_params[movie_idx,:],user_params[:,user_idx])
                  error = real_rating - predict_rating
                  movie_params[movie_idx,:] = movie_params[movie_idx,:]  + 0.001*(error*user_params[:,user_idx]
                                                                          - 0.2*movie_params[movie_idx,:])
                  user_params[:,user_idx] = user_params[:,user_idx] + 0.001*(error*movie_params[movie_idx,:]
                                                                      - 0.2*user_params[:,user_idx])
              prediction_matrix = np.dot(movie_params,user_params)
              curr_rmse = rmse(prediction_matrix, matrix_ratings)
              #print(curr_rmse)
              rmse_ls.append(curr_rmse)
```



IV. Once the parameters have been trained, we can use them to make predictions. We reshape the dataset so that each user corresponds to a list of the movie IDs for the 20 recommended movies with the best predicted ratings. The final output is a matrix with 610 rows (representing the users) and 21 columns (20 movie IDs for the recommended movies for each user, plus a column for the user ID).

In [380]:
```python
for i in range(610):
    curr_recommended_movie = {}
    # print(type(np.where(matrix_ratings.iloc[:,i] == 0)[0]))
    final_predict_ratings.iloc[np.where(matrix_ratings.iloc[:,i] != 0)[0],i] = 0
    predicted_user_ratings = final_predict_ratings.iloc[:,i]
    # print(predicted_user_ratings.nlargest(20))
    recommended_movie_index = predicted_user_ratings.nlargest(20).index.values
    curr_recommended_movie['user'] = user_list[i]
    curr_recommended_movie['recommended_movies_list'] = recommended_movie_index
    recommended_movie_df = recommended_movie_df.append(curr_recommended_movie,ignore_index = True)
```

In [385]:
```python
final_recommendation_df
```

Out[385]:

	user	movieId1	movieId2	movieId3	movieId4	movieId5	movieId6	movieId7	movieId8	movieId9	...	movieId11	movieId12	movieId13	movieId14	mov
0	1	160573	53453	165947	123200	1987	30890	25952	6342	3834	...	117364	6064	131749	86668	
1	2	53453	160573	123200	6912	30846	102684	58975	49917	165947	...	86668	40412	123310	126577	
2	3	6858	2623	7988	160573	72424	3834	141646	7282	153408	...	60538	102684	31952	33132	
3	4	160573	53453	165947	123200	1987	30890	90384	6342	25952	...	6064	107069	117364	123310	
4	5	53453	160573	4251	6382	1987	109971	165947	38583	311	...	7615	2632	7345	3834	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
605	606	160573	53453	165947	123200	1987	25952	6342	6912	90384	...	3834	6064	131749	2632	
606	607	160573	53453	165947	123200	1987	30890	25952	3834	117364	...	6912	90384	6064	311	
607	608	160573	53453	165947	123200	1987	30890	25952	6912	6342	...	6064	3834	117364	311	
608	609	53453	160573	6912	81535	123200	116138	4251	5391	165947	...	89939	64278	123310	49917	
609	610	160573	53453	165947	123200	1987	30890	25952	6342	6912	...	117364	90384	311	6064	

610 rows × 21 columns

In [386]:
```python
final_recommendation_df.to_csv('/Users/juli/Desktop/FA21/CS_410/Recommder_System/recommendation_result1.csv')
```