Movie Recommendation System using NCF

Group 22

Aditya Jain (aj780)

Parth Hasmukh Jain(pj269)

Jenil Ashwin Jain(jj822)

Problem Statement

- We aim to build a movie Recommendation System based on the Movie Lens dataset
- Given a user_id, movie_id pair, the goal is to predict the rating a user would give to a movie and to predict the top 10 movie recommendations for the user
- We make evaluation based on RMSE, Precision, Recall and NDCG

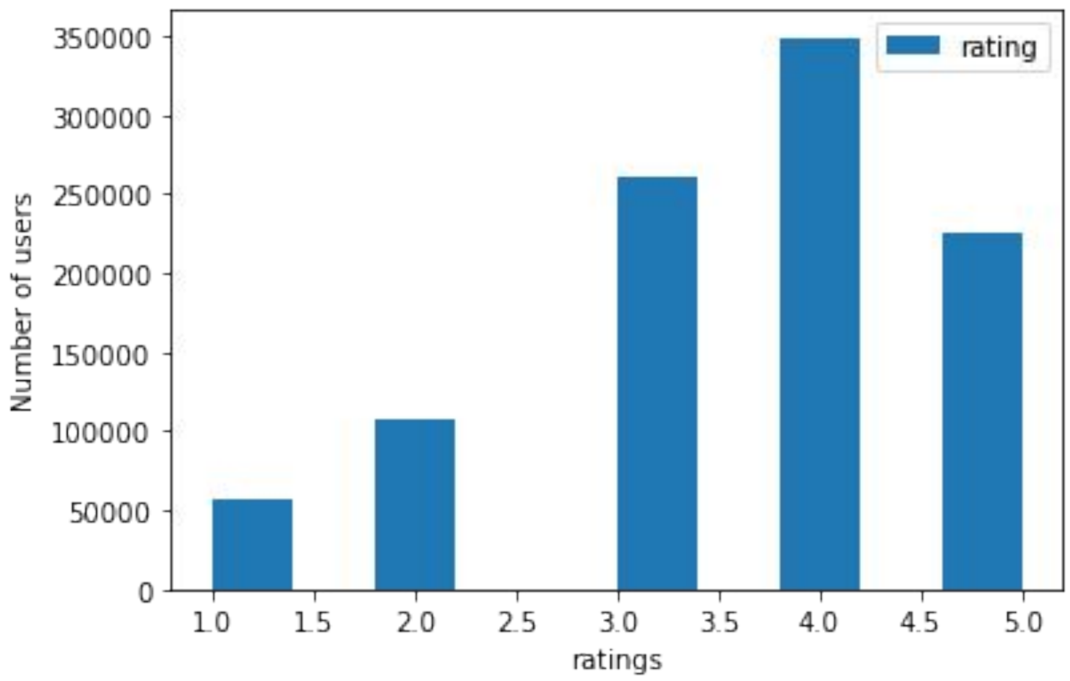
Dataset

- MovieLens review dataset (ml-latest)
  - Ratings: 1M
  - Movies: 4k
  - Users: 6k

	userID	itemID	rating	timestamp
0	1	1193	5.0	978300760
1	1	661	3.0	978302109
2	1	914	3.0	978301968
3	1	3408	4.0	978300275
4	1	2355	5.0	978824291

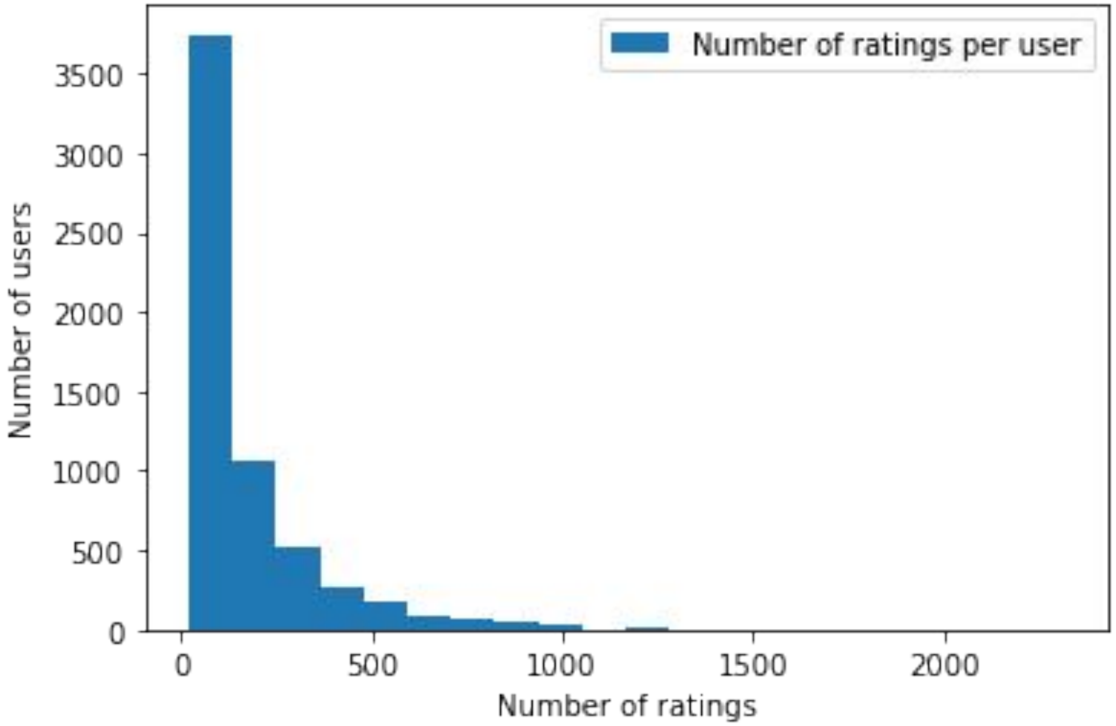
Data Analysis

Histogram of Ratings



Data Analysis

Number of ratings per user



Dataset Preprocessing & Train/Test split

- Shuffled the dataset to avoid overfitting
- Converted the user_id and movie_id to integers in order to feed it to the embedding layers of the model
- Performed a chronological 80/20 train-test split. Out of the training set 10% of the data was used for validation

Methodology Overview

Rating

Features used: **user_id, movie_id, rating**

Model used: **Neural Collaborative Filtering**

Loss function: **Mean Square Error**

Accuracy metric: **Root Mean Square Error, Precision, Recall, NDCG**

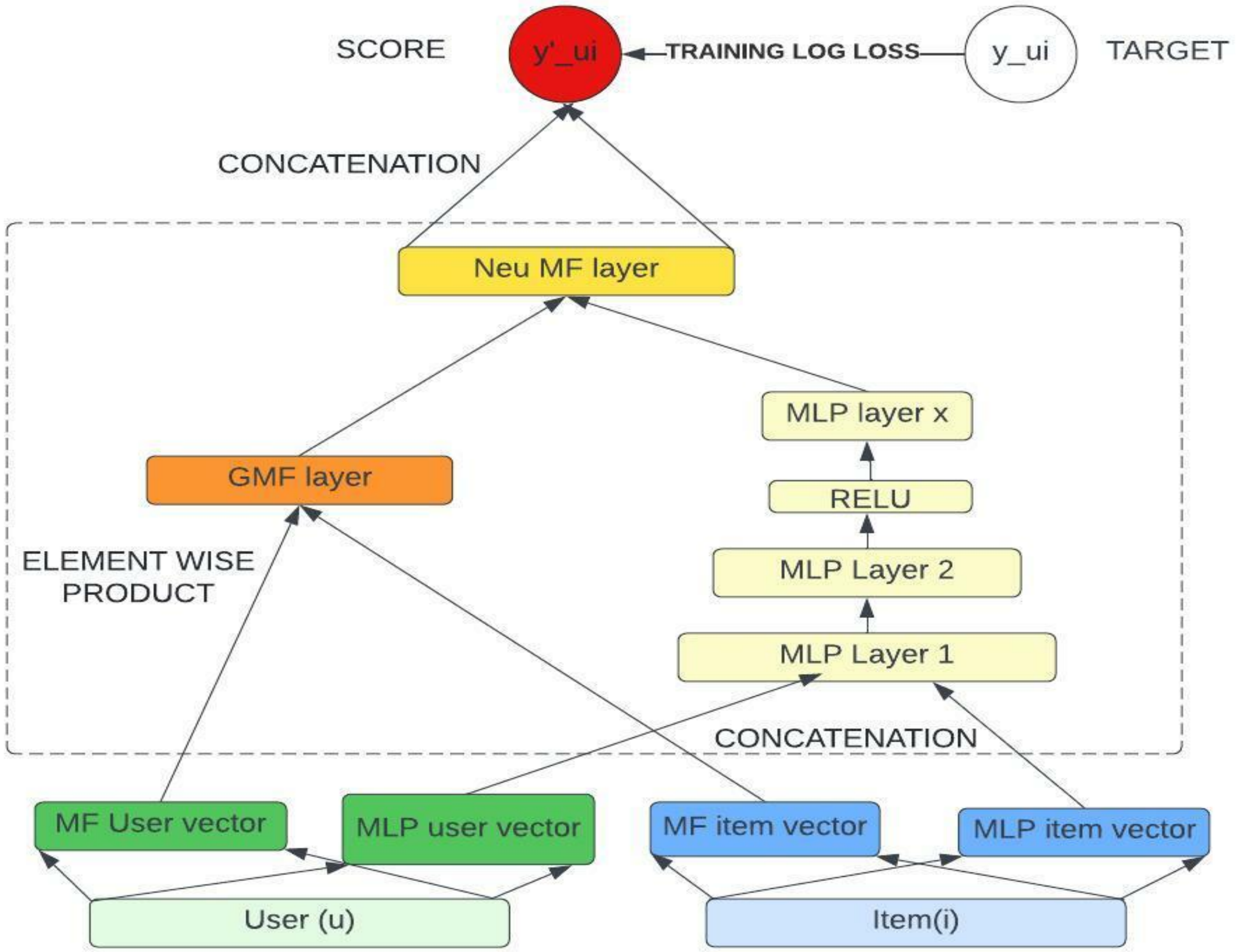
Why NCF ?

- Other methods like matrix factorization's performance is hindered by the simple choice of interaction function - inner product.

- NCF utilizes Multi-Layer Perceptron model (MLP) to learn user-item interactions

- Generalizing and expressing MF as a special case of NCF

Model Architecture



Model Summary

```
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 user_input (InputLayer)        [(None, 1)]          0           []                               
                                                                                                  
 product_input (InputLayer)     [(None, 1)]          0           []                               
                                                                                                  
 embedding_30 (Embedding)       (None, 1, 32)        193280      ['user_input[0][0]']             
                                                                                                  
 embedding_31 (Embedding)       (None, 1, 32)        118592      ['product_input[0][0]']          
                                                                                                  
 embedding_28 (Embedding)       (None, 1, 32)        193280      ['user_input[0][0]']             
                                                                                                  
 embedding_29 (Embedding)       (None, 1, 32)        118592      ['product_input[0][0]']          
                                                                                                  
 flatten_30 (Flatten)           (None, 32)           0           ['embedding_30[0][0]']           
                                                                                                  
 flatten_31 (Flatten)           (None, 32)           0           ['embedding_31[0][0]']           
                                                                                                  
 flatten_28 (Flatten)           (None, 32)           0           ['embedding_28[0][0]']           
                                                                                                  
 flatten_29 (Flatten)           (None, 32)           0           ['embedding_29[0][0]']           
                                                                                                  
 concatenate_14 (Concatenate)   (None, 64)           0           ['flatten_30[0][0]',             
                                                                  'flatten_31[0][0]']             
                                                                                                  
 multiply_7 (Multiply)          (None, 32)           0           ['flatten_28[0][0]',             
                                                                  'flatten_29[0][0]']             
                                                                                                  
 dense_13 (Dense)               (None, 64)           4160        ['concatenate_14[0][0]']         
                                                                                                  
 concatenate_15 (Concatenate)   (None, 96)           0           ['multiply_7[0][0]',             
                                                                  'dense_13[0][0]']               
                                                                                                  
 dense_14 (Dense)               (None, 1)            97          ['concatenate_15[0][0]']         
                                                                                                  
==================================================================================================
Total params: 628,001
Trainable params: 628,001
Non-trainable params: 0
__________________________________________________________________________________________________
```

Comparison with baseline models

- We used three other models to compare our NCF model
  - Singular Value Decomposition model
  - The SVD plus plus model
  - K nearest neighbor model

Results from baseline model

Model	RMSE	RECALL	PRECISION	NDCG
SVD	0.8736	0.239	0.6300	0.72
SVD plus plus	0.8676	0.2512	0.648	0.72
KNN	0.9296	0.244511	0.6411	0.71

Results using NCF

1		no of factors	no of hidden layers	rmse	precision	recall	ndcg
2	0	8	1	0.85722948	0.61361032	0.35473116	0.893112031
3	1	8	2	0.86369382	0.64042902	0.39390078	0.893815647
4	2	8	3	0.85706142	0.64041371	0.38342682	0.894368388
5	3	16	1	0.85967642	0.61494613	0.35408325	0.893637209
6	4	16	2	0.85169369	0.58466914	0.31761629	0.895376639
7	5	16	3	0.85207007	0.59376314	0.32608606	0.894195696
8	6	32	1	0.86002881	0.61063367	0.35317673	0.893381155
9	7	32	2	0.85629365	0.63403724	0.37366009	0.894659576
10	8	32	3	0.85395078	0.60217249	0.34024898	0.894762132
11	9	64	1	0.85899143	0.61368601	0.35286318	0.89272187
12	10	64	2	0.85484338	0.60478023	0.3496602	0.894535897
13	11	64	3	0.85738993	0.61959904	0.36343007	0.894483656
14	12	100	1	0.8583463	0.60732642	0.34815741	0.89250146
15	13	100	2	0.85472379	0.61417803	0.35527467	0.893688709
16	14	100	3	0.85120168	0.61522719	0.34747834	0.895174548

Future Scope

- To improve the model's accuracy, use non-numerical features like movie reviews
- To combat the issue of cold start, use user segmentation and classify user in one of the segments and recommend products in that segment.
- Use deep learning to predict time-aware recommendations and learning to rank
- Vary hyperparameters such as learning rate and adding regularization tricks like dropout to avoid overfitting

Thank you!