COLLABORATIVE FILTERING

Report Submitted By:

TEAM: PSG-IASD

Kanupriya Jain
Othman Hicheur
Nan An

Contents

1 Introduction

The project focuses on building a recommender system with a matrix by using collaborative filtering. Our goal on this project is to compute a full matrix of ratings from users and movies with only a few part of the full matrix.

Methods Implemented: We first set our baseline with the Alternating Least Squares (ALS) [2], Matrix Factorization [1] and then we employ Neural Collaborative Filtering [4] and Probabilistic Matrix Factorization [3].

2 ALS Matrix Factorization

Alternating Least Squares (ALS) is a matrix factorization method used to decompose the user-item interaction matrix into two lower-dimensional matrices [2]. The goal is to approximate the original matrix, $R \approx UI^T$, where U and I are latent factor matrices for users and items, respectively.

This library merges the *preference (p)* for an item with the *confidence (c)* we have for that preference.

Preference is set as follows [2] : $p_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 0 \\ 0 & \text{if } r_{ui} = 0 \end{cases}$

Confidence is set as follows [2]: $c_{ui} = 1 + \alpha r_{ui}$

where r_{ui} is the rating. The rate at which our confidence increases is set through a linear scaling factor α. It's default value is 1.

Loss Function:

We minimize the following loss function [2]-

$$L(U, I) = \sum_{(u,I) \in K} c_{ui} \cdot (p_{ui} - U_u^T I_i)^2 + \lambda \left(||U_u||^2 + ||I_i||^2 \right)$$

where K is the set of observed ratings.

Results: The code for the following results can be found in the jupyter notebook named `Code3_ Alternating_Least_Squares(ALS).ipynb` By using the above-mentioned library and after hyper tuning on `factors`, `iterations` and `regularization`, our best parameters were -

- `factors` = 10
- `iterations`=100
- `regularization`=0.1

Due to limited computational resources, we used a small parameter grid, which led to suboptimal results. Expanding the parameter grid could improve the performance. Performance metrics on the test data provided -

- Mean Squared Error: 12.046
- Root Mean Squared Error: 3.470

3 Probabilistic Matrix Factorization:

This method takes into account Bayesian learning for parameter learning.The R matrix in this case can be estimated by using two low-rank matrices U and V as well [3]. In this case, matrix is represented as $R \approx U^T V$. Our goal is to maximize this likelihood function [3]:

$$p(\mathbf{U},\mathbf{V}|R,\sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M} N(R_{ij};\mathbf{u}_i^\top \mathbf{v}_j,\sigma^2)\prod_{i=1}^{N} N(\mathbf{u}_i|0,\sigma_U^2 I)\prod_{j=1}^{M} N(\mathbf{v}_j|0,\sigma_V^2 I)$$

N= number of users M= number of items
Finally, by introducing some additional notation to identify the hyperparameters of the model, we will get loss function where we consider only observed ratings [3]:

$$L = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M}(R_{ij} - U_i^T V_j)^2 I_{(i,j)\in\Omega_{R_{ij}}} + \lambda_U||U_i||_F^2 ro + \lambda_V||V_j||_{Fro}^2$$

Results:

The code for the following results can be found in the jupyter notebook named `Code2_Probabilistic_Matrix_Factorization(PMF).ipynb`. We hypertuned on `lambda_U`,`lambda_V` and `n_dim`. Our best parameters are:

- `lambda_U`=0.5
- `lambda_V`=0.1
- `n_dim`=5

Root mean squared error on the test data provided = 1.035

Note: Due to limited computational resources, we used a small parameter grid, which led to suboptimal results. We implemented the PMF method and NCF approach simultaneously. Since, we were getting better results with NCF approach, we made further improvements in that method due to lack of time.

4 Neural Collaborative Filtering

4.1 Baseline Model

The Baseline model concatenates the user and item inputs. The model uses three dense layers with sizes 256, 128, 32 and the Adam optimizer , ReLU activation functions, while the output layer uses a linear activation[4].

RMSE achieved by baseline model is **1.6669**.

Note: Note that the code for all the following results of this NCF approach can be found in the jupyter notebook named `Code1_Neural Collaborative Filtering Model.ipynb`

4.2 Model Optimization

4.2.1 Cross Validation for Best Neurons Combination

We split the training data into 5 folds, where 1 fold is used for validation and 4 folds for training. we leading to 18 combinations in total (2 * 3 * 3 = 18). As shown in Figure 2, the results are very close across different configurations. In fact, running the model multiple times often results in slightly different outcomes.

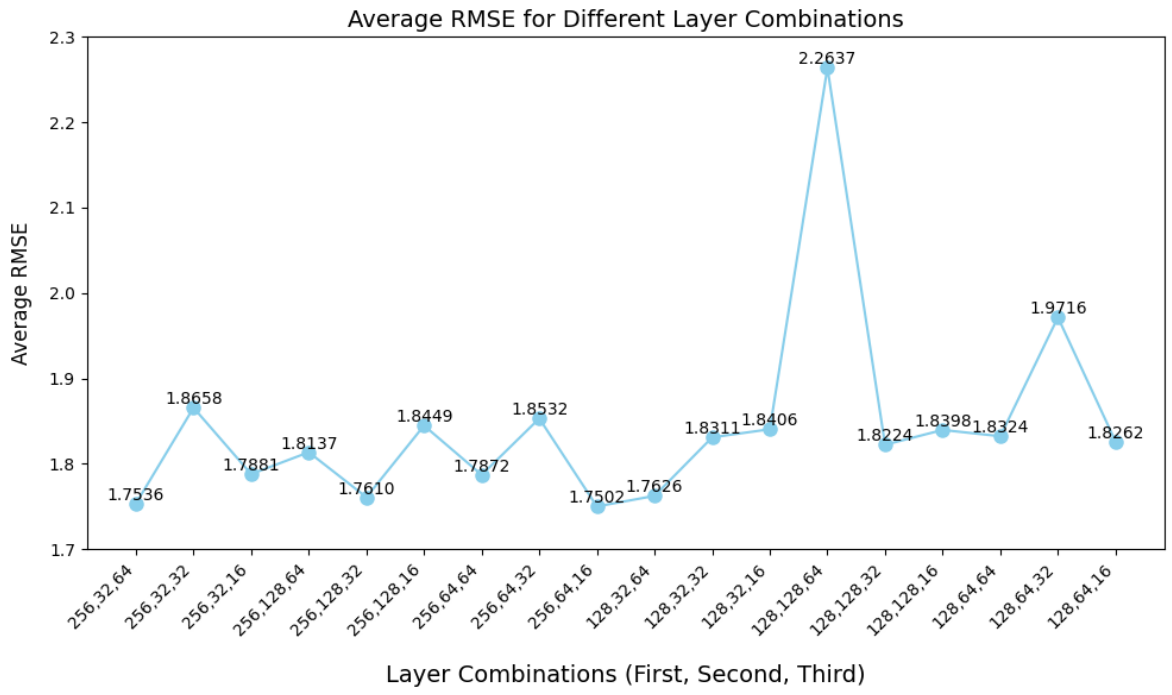

Figure 1: Cross Validation Results for Neurons Combinations

We selected the configuration with **256, 128, 32** neurons for our model.

4.2.2 Choosing the Output Layer Activation and Number of Layers

In this task, **Linear Activation** works best for our model since it fits continuous predictions. **Sigmoid** activation limits the output to a specific range, between 0 and 1. **Softmax** is commonly used in classification tasks.
With 4 layers, the RMSE was **1.6859**, with 3 layers: **1.6763**, and with 2 layers: **1.5388**. We chose 3 layers since the performance is close to that of 4 layers, and we wanted to avoid overfitting.

4.2.3 Introducing Embedding

We introduced an Embedding layer to transform user and item inputs into a 32-dimensional dense vector, which adds non-linearity and helps capture latent factors. This yielded an RMSE of **0.8873**, a significant improvement.

4.2.4 Introducing the Attention Layer

Adding an attention layer [5] did not improve performance; the RMSE increased from **0.8871** to **0.8908**.

4.2.5 Introducing Bias

Adding bias allowed the model to account for global rating tendencies, reducing the RMSE to **0.8805**.

4.2.6 Choosing the Embedding Dimension and Loss Function

When the dimension was set to 64, the RMSE was **0.8915**; for 32, it was **0.8870**; and for 16, it was **0.8956**. We chose a balanced embedding dimension of 32 to avoid overfitting while still capturing sufficient complexity.
With **MAE**, the RMSE was **0.9003**; with **MSE**, it was **0.8871**; and with **Huber**, it was **0.9033**. We selected MSE as our loss function since it penalizes larger errors more heavily.

4.2.7 Final Model

This is the process for building our model, with red lines indicating the model-building process and numbers representing the RMSE values.

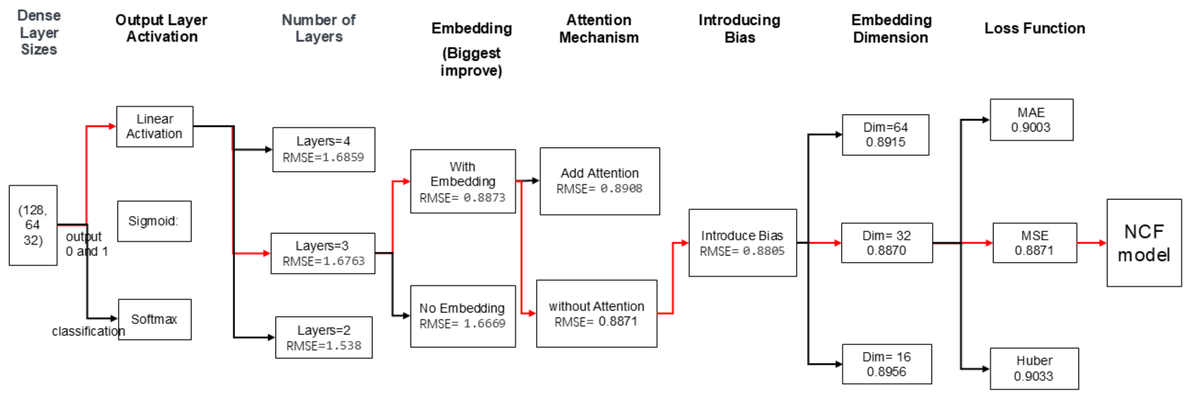

Figure 2: Decision Diagram for building the NCF (Neural Collaborative Filtering) Model

4.3 Conclusion

The experiments demonstrate that by introducing embeddings to map users and items into dense vector spaces, the model significantly improves its ability to predict user-item interactions by capturing latent factors. Accounting for systematic biases relative to the global average also helps fine-tune the model. The increase in RMSE after adding attention suggests that the attention mechanism may introduce noise by overemphasizing less relevant features.

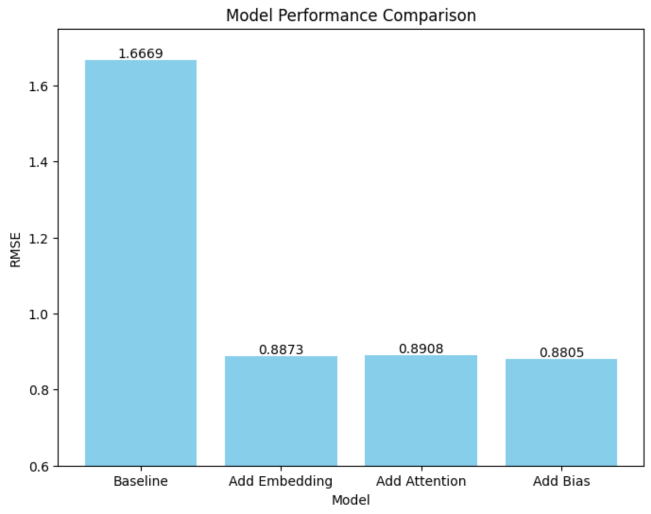

Figure 3: Model RMSE Comparison

5 Pre-processing : add of extra feature genre of movie

After extracting all the genres which are in the dataset, we create a matrix named $moviegenreencoding$. This matrix is in $\mathbb{R}^{M \times G}$ where M is the number of movies and G the number of genres. Moreover, in order to not create order between the genres, we decided to create **one-hot encoded** vector for every movie.

Now, we can include the genres of movie in the input of our model. To do that, we only have to add the genres as an input of size G, and to embed them with an activation function $ReLU$. Finally, we just add the embedding to the concatenation of the users embedding and items embedding.

Results: Our best parameters were :

- embedding dimension = 32
- global average of rates = 3.5246 (the mean of the rates on training set)
- λ parameter of regularization = 10^{-5}
- batch size = 256
- number of epochs = 5

Root mean squared on the test data provided = 0.879

6 Post-processing : potential use of the structure of the real ratings

After working on the creation of the best model, and thinking on the best way to pre-process the initial data, we tried to find the best way to post-process our predictions. It is important to notice that all the real values of ratings are in the following space : $\mathbb{V} = \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0\}$. So, the main idea of our work on the post-processing is to find the best mapping between the predictions that belong to $\mathbb{R}$ and the real values that are on $\mathbb{V}$. Here are all our attempts to compute this idea :

- Round the predicted rate to the nearest element of $\mathbb{V}$.
- As it is shown in the previous figure : We can see that the ratings are mostly semi-integers. So, we can

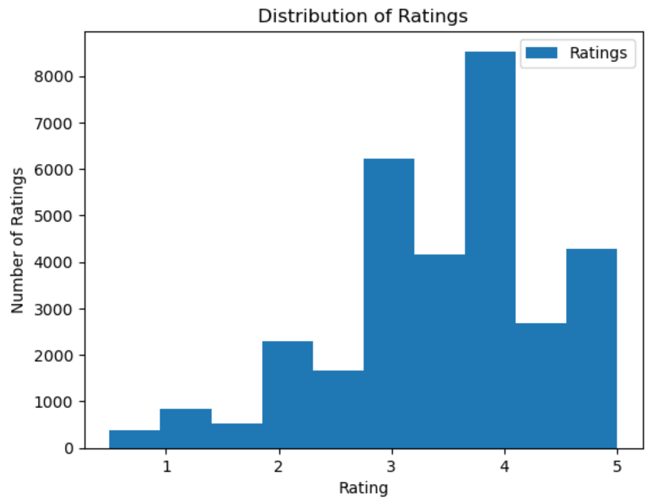

Figure 4: Histogram of the ratings in the training set

  round the prediction to an integer n of $\mathbb{V}$ if the absolute difference between n and the prediction is less than ϵ.
- Use a classifier to classify our predictions on their good class, where the class are the 10 elements of $\mathbb{V}$.
- Find the best function $f : \mathbb{R} \to \mathbb{V}$ which gives the best mapping on the training set and uses it on the predictions that are made from the test set. This approach for the post-processing is, intuitively, good because it uses the distribution of the rateings on the training set.

References

[1] Recommender System — Matrix Factorization

[2] ALS Implicit Collaborative Filtering

[3] PMF for Recommender Systems

[4] Neural Collaborative Filtering

[5] Attention Is All You Need