User-based Recommender System using Matrix Factorization and ANN

Firas Jolha
f.jolha@innopolis.university
May. 2021

Course: Advanced Machine Learning
Project Github Link: https://github.com/firas-jolha/hw2

Motivation

One of the benefits of AI is to help people in decision making. To make good decisions, the user of the system needs recommendations, these recommendations should be data-driven and unbiased. A lot of researchers have conducted experiments to improve the recommendations with different objectives. It is common that the most powerful learning algorithm in machine learning is ANN(Artificial Neural Network). In this work we investigate and implement two algorithms for building recommendations (in fact, building the full rating matrix). The first algorithm is Matrix Factorization which could be trained using ALS(Alternating Least Squares). The second is a deep neural network which is called neural collaborative filtering.

Our recommender system is applied on a dataset of movies and users, the objective is to build the full rating matrix. The report is divided into sections as follows:

Description

In this work, we implemented a user-based Recommender system using collaborative filtering. The system takes some ratings as training data and tries to build the full rating matrix in which every user has a rating for each movie (item). We used two collaborative filtering methods but the data preprocessing steps are shared between them. In this section, we will explain the stages of data preprocessing and the implemented methods of collaborative filtering. The results will be discussed in the next section.

1. Data Description and Preprocessing

The input to our recommender system is user ID and the output is a list of movies with ratings for that user.
Our dataset consists of users and movies with ratings. The training set has 6687 users and 5064 movies and 761972 ratings. It is clear that we don't have the rating of every user to every movie. Our objective in this project is to fully build this ratings matrix. A sample of training data is shown in table 1. If we do a simple analysis on the values of user ids and movie ids, we see that the ids have different scales and don't start from zero. To overcome this issue we remap or reset the ids to the basic indexing such that ids of users and movies are in a range [0, n-1], where n is the number of users and movies respectively.

	userId	**movieId**	**rating**
0	1	32	3.5
1	1	47	3.5
2	1	50	3.5
3	1	253	4.0
4	1	260	4.0

Table 1: A sample of training data

We use the new values of user and movie ids to build the rating matrix. We use a sparse matrix coo_matrix from scipy library to define the rating matrix for fast computation.
The stages of data preprocessing are:

A. **Read the data.**
B. **Create the mapping array and persist it locally.**
C. **Map the ids of users and movies (Reset the ids).**
D. **Build the sparse ratings matrices and persist it for later use.**

2. Matrix Factorization

Matrix factorization is a method for decomposing the ratings matrix R into the multiplication of two matrices P and Q such that $R_hat\ =\ P\ @\ Q.T$such that $R_hat$is an approximate rating matrix built by matrix factorization.

2.1. Basic Collaborative Filtering using Alternating Least Squares

The idea in this algorithm is to learn the matrices P and Q from given ratings by gradually adjusting their values according to the gradient of the objective function which should be minimized. This method works by alternating the procedure of adjusting P and Q matrices. That means when we adjust P, we consider Q as fixed and vice versa. This facilitates the computation of gradients that are used to update the values in the matrices. If the number of users and items in the system are m and n respectively and let $k$be the factorization rank then P has the shape $m\ x\ k$ and Q has $n\ x\ k$ such that $R\_hat\ =\ P\ @\ Q.T$

The stages of training in this method are as follows:

1. Take sparse rating matrix R from preprocessing step.
2. Initialize the values of P and Q randomly sampled from uniform distribution so that the values are in the range [0-1)
3. Compute the gradient of the loss function for P and Q separately (by fixing one of them and calculating the gradient according to the other).
4. Update P values using the gradient and other hyperparameters like learning rate and regularization rate.
5. Update Q values using the gradient similarly to the previous step.
6. Calculate the new loss for tracking the progress.
7. Repeat steps 3 to 6 until convergence (for a number of epochs or until the loss becomes very small).
8. Our model consists of the matrices P and Q so we persist them for later use.

The stages of testing are:

1. Load the matrices P and Q from disk.
2. Load the sparse ratings matrix for test data.
3. Calculate the loss using P, Q and rating matrix.

2.2. Neural Collaborative Filtering (NCA)

In this method, we train a neural network which consists of two embedding layers for users and items followed by a stack of fully connected layers, the last layer is a single output linear layer in which the output is mapped to the ratings range [1-5]. Further description is provided in the next section.

Results and Discussion

In this section, we will show the obtained results and discuss more on them. First we will present the results of matrix factorization which we denote it as basic collaborative filtering, then the results of neural collaborative filtering will be presented.

1. Basic Collaborative Filtering (Matrix Factorization using ALS)

The configurations in the experiments are shown in table 2. The values in this table are chosen empirically after a series of experiments.

Hyperparameters	**Value**
# Users	6687
# Items	5064
Latent space dimension	7
Learning Rate	1e-6
Regularization Rate	1e-12
Epochs	30

Table 2 : Configurations of the experiments for basic collaborative filtering method

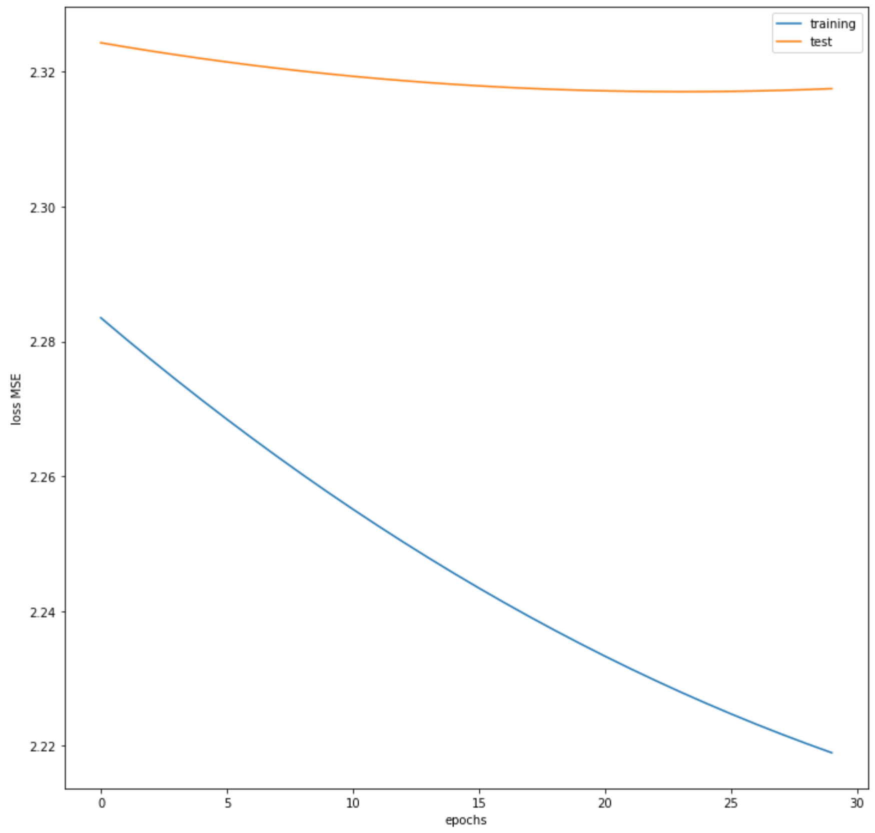

Figure 1: Training loss vs. Test loss for basic collaborative filtering

The figure 1 shows the decreasing loss for both training and test dataset trained for 30 iterations. It is clear that this model is underfitting. The high learning rate increases the step toward the minimum value in the objective function and makes so increases the cost. The low learning rate slows down the learning procedure.

The latent space dimension has an impact on the loss such that increasing the dimension will increase the loss, but decreasing it will let the algorithm to give up learning early.
It is better to increase the epochs, as it is shown in the figure 1 that the next epochs could decrease the loss more.

The average loss for test data in case of basic collaborative filtering was `2.2416245`

2. Neural Collaborative Filtering

This method utilizes a neural network for learning the latent space of the rating matrix. The neural network mainly consists of two embedding layers for both users and items followed by a stack of fully connected layers. The output layer is a single linear layer with Sigmoid as an activation function. The output range of Sigmoid is [0-1], but the ratings range is [1-5] so we scale the output of the Sigmoid function to be in the specified range. The input to the embedding layers is a list of user and item ids but we could do one-hot encoding for them or not as the ids are in nominal scale. We have two options for building this neural network, with one-hot encoding preprocessing applied on the input features or without it.

The configurations of experiments on neural collaborative filtering are as follows:

Hyperparameters	**Value**
# Users	6687
# Items	5064
Latent space dimension	7
# neurons of Fully connected layers	[-1, 64, 16, 8, 4]
Learning Rate	1e-3
Loss	MSE
Epochs	10

Note: The in_features for the first FC layer is -1 because it differs from the model with one-hot encoding than without it.

a. With On-hot Encoding

In this case, our model is a neural network and has the following schema.

```
NCA(
  (embed_user): Embedding(6687, 7)
  (embed_item): Embedding(5064, 7)
  (fc_layers): ModuleList(
    (0): Linear(in_features=82257, out_features=64, bias=True)
    (1): Linear(in_features=64, out_features=16, bias=True)
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): Linear(in_features=8, out_features=4, bias=True)
  )
  (dropout): Dropout(p=0.2, inplace=False)
  (output): Linear(in_features=4, out_features=1, bias=True)
  (output_f): Sigmoid()
)
```

This model has 5347986 trainable parameters and takes so much time in training each epoch. The training loss for this model is shown in figure 2 which is run for 10 epochs.

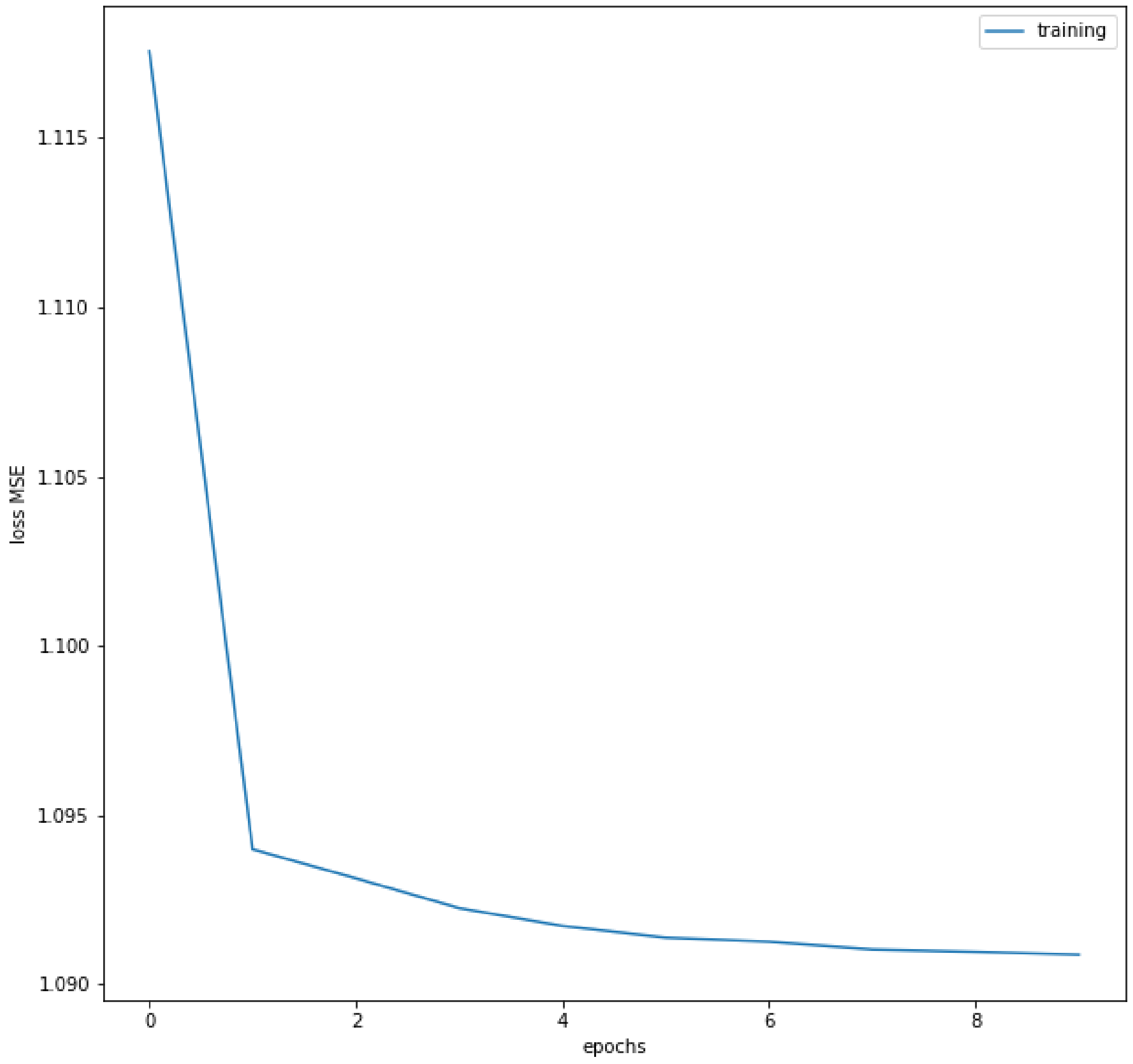

Figure 2: The training loss of the neural collaborative filtering with one-hot encoding

The average test loss is 1.0913212421364809

b. Without One-hot Encoding

The model has the following schema in which the input is only two features user ids and item ids. This model has 84434 trainable parameters and is faster in training than the previous model for each epoch.

```
NCA(
  (embed_user): Embedding(6687, 7)
  (embed_item): Embedding(5064, 7)
  (fc_layers): ModuleList(
    (0): Linear(in_features=14, out_features=64, bias=True)
    (1): Linear(in_features=64, out_features=16, bias=True)
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): Linear(in_features=8, out_features=4, bias=True)
  )
  (dropout): Dropout(p=0.2, inplace=False)
  (output): Linear(in_features=4, out_features=1, bias=True)
  (output_f): Sigmoid()
)
```

The training loss in this case is shown in figure 3.

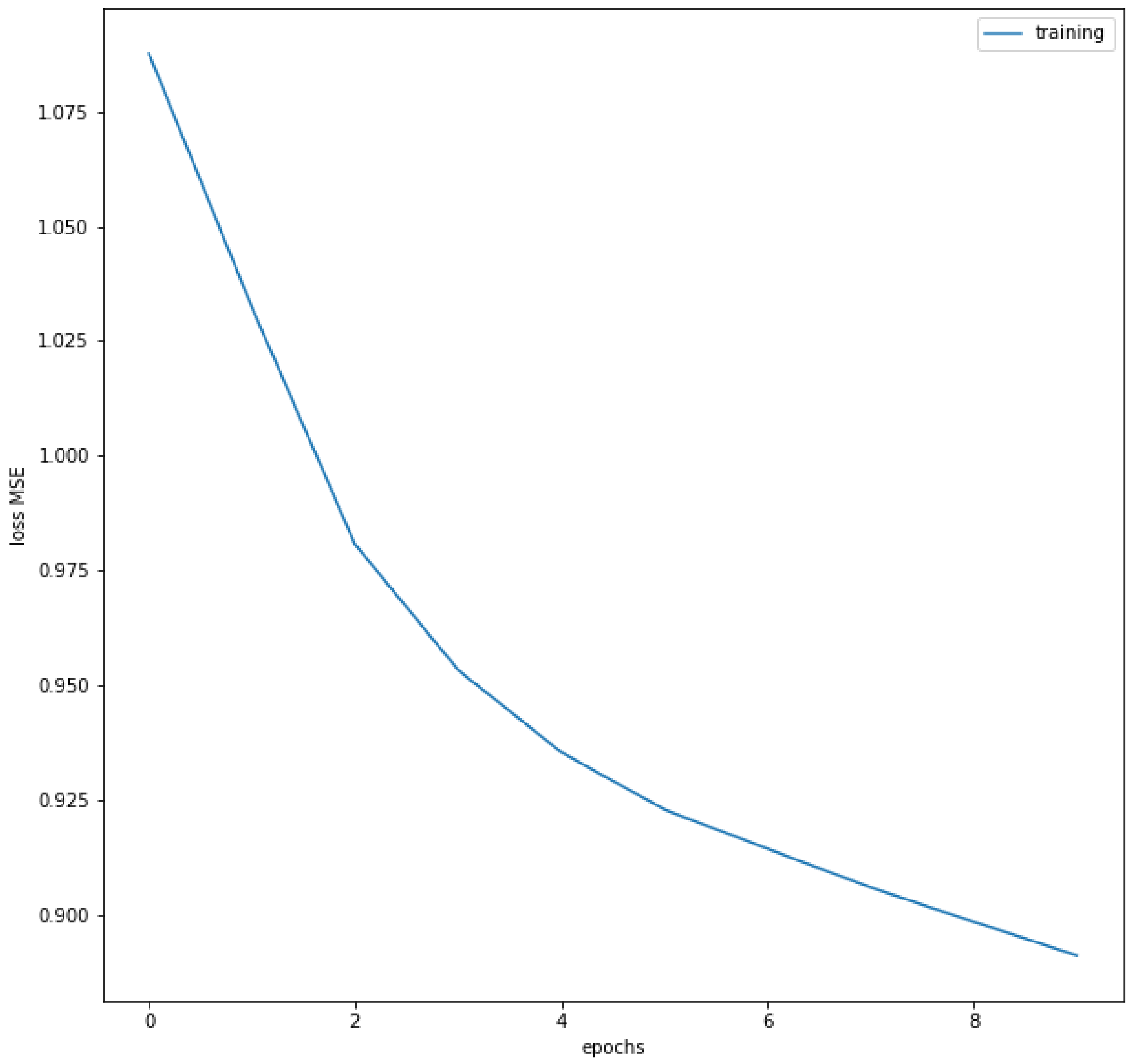

Figure 3: The training loss of the neural collaborative filtering without one-hot encoding

The average loss for test data in this model is `0.9069439294`.

3. Comparison

It is clear that neural CF methods outperforms basic CF method. The next table shows the results of comparison between three models (basic CF, neural CF with one-hot encoding and neural Cf without one-hot encoding).

Criteria	Basic CF	Neural CF + One-hot	**Neural CF**

		Encoding	
# Params	82,257	5,347,986	**84,434**
Average Tes Loss	2.2416245	1.0913212421364809	**0.9069439294**

Neural CF outperforms both Neural CF with one-hot encoding and basic CF.

Generating Recommendations

For generating recommendations, the user id is required then a list of top items (movies) for that user will be retrieved with ratings. The steps of generating recommendations are:

1. Let $u$to be a user id.
2. Map u to the basic indexing and we get u'
3. Calculate $R\_hat = P @ Q.T$
4. Get the ratings of the user u' from R_hat
5. Sort the ratings
6. Keep only top N ratings and their indices, the result is a list of item ids with ratings.
7. Remap the item ids to their original index using helper functions from preprocessing steps.
8. The result is a dataframe of two columns (item ids and ratings)

Note: If the user id $u$is not found in the ratings matrix, then an error message will be printed.

The following table presents a sample of recommendation results.

```
Recommendations for User : 6321
   Item   Rating
0   6870  2.420039
1  67923  2.409719
2  60753  2.392414
3   5246  2.362057
4   2451  2.333585
5   1888  2.325671
6   2054  2.316188
7  34530  2.308407
8    457  2.306284
9    535  2.296287

Recommendations for User : 11231232
Error: No users like this
None

Recommendations for User : 1
   Item   Rating
```

```
0   5246  4.115012
1   8800  4.036552
2   1230  3.762852
3   3285  3.750736
4   3705  3.722233
5  47629  3.714306
6   7324  3.709528
7   2143  3.708638
8   3165  3.696657
9    535  3.689434
```

Used Tools and Technologies

Google Colab, Python, Atom editor, pdoc tool, Github and Pytorch.

Conclusion

This assignment was a good practice on designing a recommender system using ALS and NCF. The additional preprocessing that is used is the one hot encoding of user and movie ids but it makes the model more complex in training and testing.
If the response time is a concern in the system then ALS is a satisfying approach but has less performance than NCF. NCF gives the opportunity to build more robust models but it is time consuming which could be improved by adding additional preprocessing steps before feeding the data to the neural network.