STA561 Final Project - Movies Recommender System

Zilin Yin

April 30 2020

1 Background

The recommender system is prevalent nowadays. People encounter various kinds of recommender systems in a variety of ways, whether they decide to see a movie at Netflix or merely want to buy a book on Amazon. These recommender systems commonly serve one purpose, which is to recommend products based on people's interest, in such a way of gaining commercial profit. Within the intense competitive atmosphere of recommender systems, a system that has outstanding recommendation performance because of its right choice of mechanism and consistent upgrading with emerging technologies would be extremely preferable, as the commercial value it could bring is enormous. Hence, a study regarding methods to improve recommender system performance could be useful.

2 Obejective

For this project, we are making a movie recommendation system with several different methodologies that will be introduced later. The purpose is to come up with a model that performs optimal rating prediction based on evaluation of the chosen scoring metrics.

3 Related Work

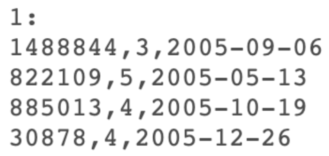
Traditionally, the methods used for recommender systems include content-based methods, collaborative filtering methods and hybrid methods which is a combination of the two. Content-based methods focus on the features of the movies themselves. In such a scenario, the model input would be a full set of movies' features including release date, genre, movie length, movie description and so on. It sometimes involves the implementation of NLP techniques. The model would then make personalized rating predictions based on the movie features (Kevin Luk, 2019).

Collaborative filtering methods, instead, operates on user-movie matrices, the entries of the matrices are the ratings given by the users to the movies. Collaborative filtering methods often involves the idea of Matrix Factorization, which is strictly connected to a common matrix decomposition algorithm called singular value decomposition (SVD). The main idea of collaborative filtering methods is to let the model figure out the latent features related to similarities, and then based on which, make personalized rating predictions (Lazy Programmer Inc.).

For this project, the focus will be on collaborative filtering methods. One early work Recommendation System for Netflix (Leidy Esperanza Molina Fernandez, 2018) explores the effect of using different collaborative filtering methods as well as different model-based algorithms on building the Netflix movie recommendation system. It has been shown that with different methods and algorithms chosen to use, the prediction performance could be varied quite a lot.

4 Dataset

We will use the Netflix official Dataset from Netflix Prize Competition. The main dataset includes 480189 users' ratings to 17770 movies, with users' IDs and movies' IDs uniquely specified. Notice that not all movies are rated by each user. The ratings are ranged from $0 - 5$, in integers. The dataset is provided in txt format as follow:



```
1:
1488844,3,2005-09-06
822109,5,2005-05-13
885013,4,2005-10-19
30878,4,2005-12-26
```

Figure 1: Netflix Prize Competition Dataset Sample

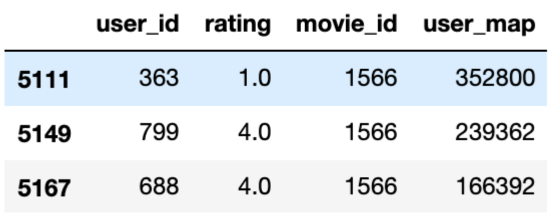
For lines in the file containing movie IDs, which are followed by a colon, the subsequent lines have the following format: Customer ID, Rating, Date of Rating (Netflix).

5 General Approach

The first step of the project is data preprocessing. As indicated in the dataset section, the size of the dataset used is quite large. Hence, it must be treated in a well manner, such that later analyzing works could be done successfully. We will also shrink the dataset to save computational cost (not for autoencoder, which will be explained later). The second step is to input the preprocessed dataset into several different collaborative filtering models that we considered to be worth giving a try, and to come up with their validation scores based on chosen scoring metrics. In the end, we will compare the results from different models and conclude.

6 Data Preprocessing

We use pandas to read in the provided txt files, and perform operations including removing the movie id rows, adding a new movie id column such that the entire data frame becomes more succinct and consistent. The resulting data frame has shape 100480507 by 3. We then shrink the data frame to shape 9623987 by 4 (there is one additional column because we add one remapped user id column) by keeping rows with most information, which to be more specific, is determined by the amounts of movies each user has rated, and the amounts of times each movie has been rated by users. Following is a sample of the preprocessed dataset:



	user_id	rating	movie_id	user_map
5111	363	1.0	1566	352800
5149	799	4.0	1566	239362
5167	688	4.0	1566	166392

Figure 2: Preprocessed Data

And it could be easily put into the models we want to use.

7 Models

7.1 Matrix Factorization (MF) Model with Keras

Matrix Factorization is about decomposing a huge, sparse matrix into two matrices that are relatively smaller in size, and the difference between the original huge matrix and the product of the decomposed matrices is expected to be as small as possible, which for the case of this study, can be represented as follow:

$$R \approx \hat{R} = WU^T \tag{1}$$

Where R and $\hat{R}$ are both N by M matrices, with N denotes the number of users and M denotes the number of movies. W is a N by K matrix and U is a M by K matrix. Notice that the K here can be interpreted as the amounts of latent features we intend to have for both W and U matrices. Since W and U are both feature matrices, we will call them user embedding and movie embedding later on to make more sense). One thing to be noticed is that R here is exactly what we want for the output of the model, which contains all users' rating to all movies. Then the following loss function is used to solve for W and U, and relevant terms:

$$J = \sum_{i,j \in \Omega} (r_{ij} - \hat{r}_{ij})^2 + (||W||_F^2 + ||U||_F^2 + ||b||_2^2 + ||c||_2^2) \tag{2}$$

Where b is a N by 1 vector denoted as the user bias term and c is a M by 1 vector denoted as the movie bias term. The μ term is used to mitigate the influence brought by over pessimistic or over optimistic users (e.g. some users give a rating of 5 if a movie is good, and 0 if a movie is bad, but never consider scores between 0 and 5). To be more specific, with the μ term, what the model is running based upon is the deviation of the ratings rather than the ratings themselves. The Ω term denotes the entries that are not empty, because we don't want to include empty entries when fitting the model. And lastly, the subsequent λ term is a l2 regularization penalty to the loss function.

In simple, the model will reproduce the original N by M matrix from a N by K user embedding and a M by K movie embedding with a specified loss function that acts only on the cells that are not empty in the original matrix.

7.2 Deep Neural Matrix Factorization (MF) with Keras

We also extend the previous model into a deep neural network model. The main difference between this model and the previous one is that after user and the movie embeddings, a N by K matrix, denoted as W, and a M by K matrix, denoted as U are initialized. The input of this model would be the flattening of the concatenation of W[i] and U[j] where i,j, rather than the embeddings themselves.

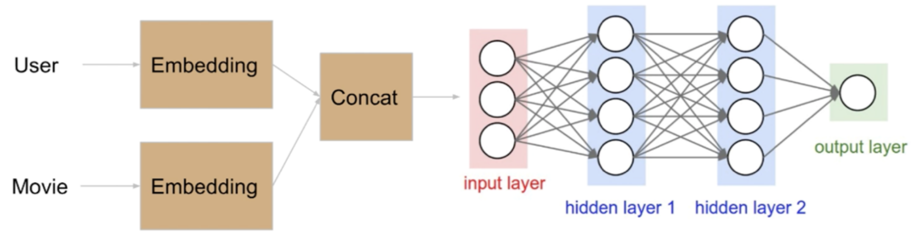
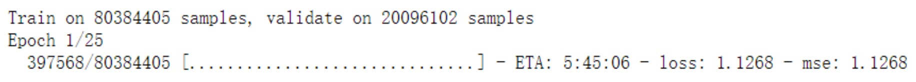


Figure 3: Graphical illustration of a matrix factorization neural network (Lazy Programmer Inc.)

By looking at only the left side of Figure 3, we see that a deep neural matrix factorization model shares some similarities with a matrix factorization model, as they both have user and movie embeddings. While this being true, the biggest difference between the two models is that a deep neural matrix factorization models includes a neural network, such that it is looking for non-linear patterns between the input and the output while a matrix factorization model is looking for linear patterns.

7.3 Denoising Autoencoders with Keras

One thing to be noticed is that for the two models mentioned above, the N by M user rating matrix is shrunk to a smaller size by randomly keeping specified number of rows and columns of the original matrix before been fed into the models. The reason of this is computation cost. Below is the time taken for one epoch roughly:



```
Train on 80384405 samples, validate on 20096102 samples
Epoch 1/25
  397568/80384405 [.............................] - ETA: 5:45:06 - loss: 1.1268 - mse: 1.1268
```

Figure 4: Computation Cost Illustration

Obviously, 5 hours for 1 epoch is not practical. If we want to make a complete prediction with the whole matrix rather to make multiple relatively small-scale predictions section by section (in other words, to save time), an alternative is to use autoencoders. Compared to the two models introduced above, autoencoders have significantly lower computation cost. When training the two models mentioned above, during each epoch (iteration), there are 100480507 samples, which is the number of cells filled in the N by M rating matrix, to loop over with. In comparison, for an autoencoder model, the number of samples to loop of with is dependent on how many users there are, which in other words, is the row number of the user rating matrix N. With this being said, the reason behind of it is the special architecture of autoencoder models, which is shown below:

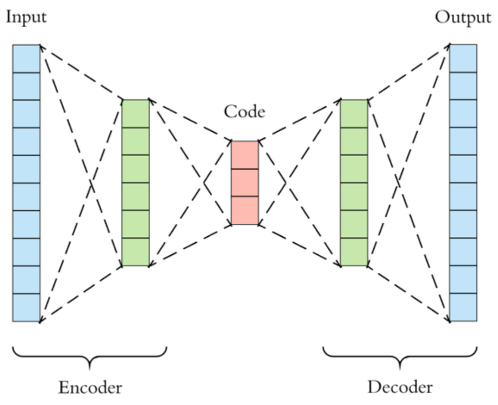


Figure 5: Autoencoder architecture graphical representation (Ardent Dertat)

The main idea here is to view each row of the user rating matrix (N by M) as a sample, which would give us N samples in total. Notice that each sample has multiple missing values, which we called noises. They are the ratings we want to predict correspondent to that specific sample user. In this case, we assign all missing cells/noises values of 0. Each sample is then fed into the model as the input, while the output of the model is set to be identical to the fed input, the loss function is specified as follow such that the model won't try to reproduce the missing cells to 0.

$$\begin{gathered} J = \frac{1}{|\Omega|}\,\Sigma_{i=1}^{N}\,\Sigma_{j=1}^{M}\,m_{ij}\,(r_{ij} - \hat{r}_{ij})^2 \\ where\ m_{ij} = 1\ if(i,j) \in \Omega\ else\ 0 \end{gathered} \tag{3}$$

Instead, the model would denoising those missing cells/noises to certain ratings.

8 Results

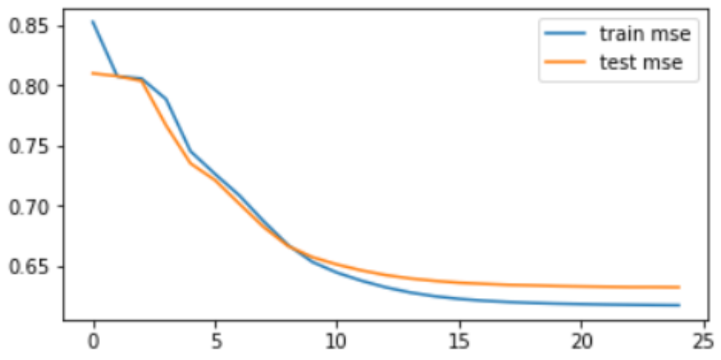


Figure 6: Matrix Factorization Model MSE

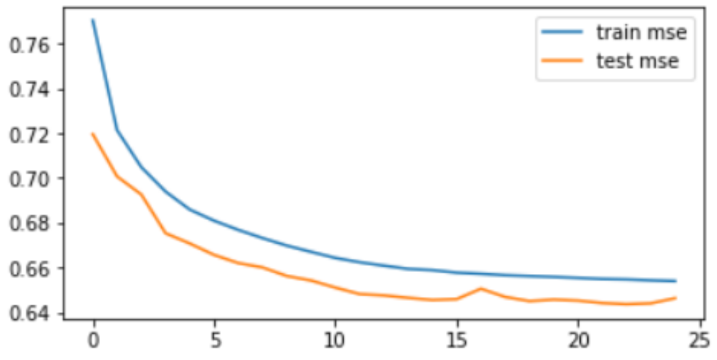


Figure 7: Deep Neural Matrix Factorization Model MSE

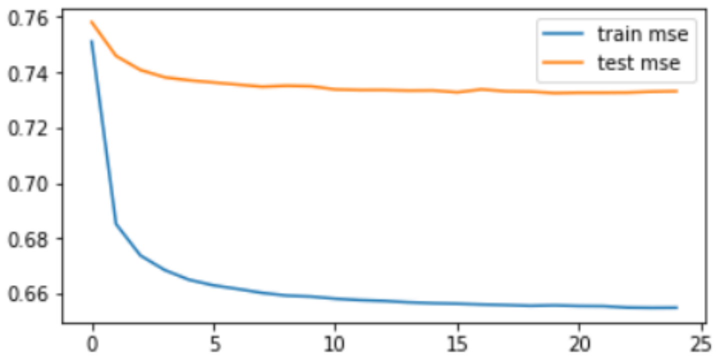


Figure 8: Denoising Autoencoder MSE

9 Discussion & Future Works

As results, the Matrix Factorization (MF) and the Deep Neural Matrix Factorization (MF) models have resulting MSE scores of around 0.65, which are better than the autoencoder model's score of around 0.73. While this being true, be attention that the autoencoder model takes in the entire user rating matrix as the input. Hence, the MSE score of the autoencoder model is a comprehensive representation (which could very likely explain why the gap between the test MSE and valid MSE of the autoencoder model is larger than the gaps of the other two) of how the model performs on the dataset. In comparison, the MF model and the deep neural MF model take only portion of the matrix as the inputs. Thus, their MSE scores, even though slightly lower than that of the autoencoder model, are not representative enough. Also, as mentioned before, the computation time (to predict the whole N by M matrix) of the autoencoder model is significantly less than the other two. Therefore, it's trade-off between using an autoencoder model for less computation cost and using MF models for higher prediction performance.

For the choice between MF model and deep neural MF model, as mentioned before, MF model tends to look for linear relationship and deep neural MF model tends to look for non-linear relationship. For the case of our study, the two models perform pretty much the same. However, for some specific dataset, depending on how its latent relationship lies inside, the choice between the two models can be very influential.

We also compare the model scores with the scores (which we refer to as baseline) in *Recommendation System for Netflix*:

	Baseline	MF	Deep Neural MF	Autoencoder
RMSE	0.6675	0.7997	0.8258	0.8562

Table 1: RMSE Scores

In the end, the three scores from the models are all lower than the baseline score. And we have concluded some possible reasons. First is that the complication of the models is still not enough. However, due to hardware limitation, we could hardly further complicate the models. Second is time limitation. Recommender system is a heat topic and currently there are tons of excellent models out there that have excellent performance. However, for now, we certainly don't have enough time to try all of them. If possible, more models could be implemented in the future, and see if they can yield better results.

10 References

Arden Dertat (2017). Applied Deep Learning-Part 3: Autoencoders.
https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Leidy Esperanza Molina Fernandez (2018). Recommendation for Netflix.
https://beta.vu.nl/nl/Images/werkstuk-fernandez$_tcm235-874624.pdf$

Kevin Luk (2018). Introduction to TWO approaches of Content-based Recommendation System.
https://towardsdatascience.com/introduction-to-two-approaches-of-content-based-recommendation-system-fc797460c18c

Lazy Programmer Inc. Recommender Systems and Deep Learning in Python.
https://lazyprogrammer.me/

Netflix. Netflix Prize.
https://netflixprize.com/index.html