CSDS 435 P3 Report Paper

Project 3

Randolph Zhao
yxz1648@case.edu

ABSTRACT

In the project, I implemented five recommendation algorithms and compared their performance and consistency using RMSE and MAE. The result shows SVD and neural network perform well but there are still many improvements that can be made to further improve the performance of these movie recommendation algorithms.

1 Introduction

In this project, I implemented five algorithms for a recommender system on the small MovieLens dataset (100K ratings): random guessing, K-Nearest Neighbors (KNN), and Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), and deep neural network. I measured and compared their performance using the Root of Mean Squared Error and Mean Absolute Error. The result shows that SVD and neural network have the best performance among these algorithms, considering RMSE, MAE, and running time. From the analysis, I believe there is still much space to improve the performance of these algorithms.

2 Methods

2.1 Random Guessing

The algorithm will calculate a normal distribution of the training set and then randomly generate ratings in this distribution as predictions. The mean and standard deviation of the normal distribution is calculated as Eq. (1) and (2), where R is all the ratings in the training set. It does not provide good predictions but can be used to compare performances with other algorithms. I implemented it using the NormalPredictor model of the Python Surprise library. Since it is randomly guessing based on the normal distribution, there is no hyperparameter needed to tune.

$$\hat{\mu} = \frac{1}{|R|}\sum_{r\in R} r \tag{1}$$

$$\hat{\sigma} = \sqrt{\frac{1}{|R|}\sum_{r\in R}(r-\hat{\mu})^2} \tag{2}$$

2.2 K-Nearest Neighbors (KNN)

For this project, since we need to estimate the rating, we will use K-Nearest Neighbor regressor. It will find the closest K data points based on some distance measurements and then use the average as the prediction. It is also a lazy-learning model. I implemented it using the KNeighborsRegressor model of the Python scikit-learn library.

In this project, I introduced the tags as features. I encoded the tag texts using the RoBERTa model, which is a pre-trained model that provides good sentence embedding vectors, and then calculate the distance matrix using Minkowski distance with p=2 as defined in Eq. (3).

$$d(u,v) = (\sum_i |u_i - v_i|^p)^{1/p} \tag{3}$$

Then, I fine-tuned the hyperparameters. The most important hyperparameter for KNN is the number of neighbor K. I split the dataset into 75%/25% and measured the coefficient of determination R^2 for K value in range from 3 to 40 as shown in Fig. 1. The result shows the best K value is K=6.

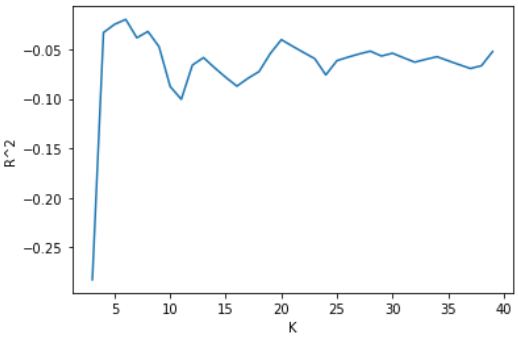

Figure 1: KNN hyperparameter tuning

2.3 Singular Value Decomposition

Singular Value Decomposition (SVD) is a popular method for dimensionality reduction or matrix factorization. Basically, SVD tries to decompose a matrix into three lower-dimensional matrices as shown in Eq. (4), where in our cases, P represents user latent factors and Q represents movie latent factors. Let r represents a rating, b represents bias, u represent the user, i represent the movie, the loss function is given as Eq. (5) and the prediction is given as Eq. (6). I implemented it using the SVD model of the Python surprise library.

$$A = PSQ^T \tag{4}$$

$$\sum_{r_{ui}\in R}(r_{ui} - \hat{r}_{ui})^2 + \lambda({b_i}^2 + {b_u}^2 + \|q_i\|^2 + \|p_u\|^2) \tag{5}$$

$$\hat{r}_{ui} = \mu + b_u + b_i + {q_i}^T p_u \tag{6}$$

For SVD, it is important to fine-tune the number of factors and the number of epochs. I set up Grid Search with 5-fold cross-validation for factors in range (5, 15) and epochs in range (50,

551). As shown in Fig. 2., the best number of factors is 7, and the larger epoch SVD has, the better. But due to time concerns, I will only stop by 550 epochs.

```
# Need to fine-tune the number of factors, and epoch number
param_grid = {'n_factors': range(5, 15, 1)}

gs = GridSearchCV(SVD, param_grid, cv=kf_sp, measures=['rmse'], n_jobs=-1)
gs.fit(ratings)

print(gs.best_score['rmse'], gs.best_params['rmse'])
✓ 8.5s
0.9829946083734262 {'n_factors': 7}


param_grid = {'n_epochs': range(50, 551, 100)}

gs = GridSearchCV(SVD, param_grid, cv=kf_sp, measures=['rmse'], n_jobs=-1)
gs.fit(ratings)

print(gs.best_score['rmse'], gs.best_params['rmse'])
✓ 4m 32.5s
0.9771587866052149 {'n_epochs': 550}
```

Figure 2: GridSearchCV for hyperparameters of SVD

2.4 Matrix Factorization

For the matrix factorization with regularization, I introduced the Non-negative Matrix Factorization (NMF). It is very similar to SVD, but it requires the decomposed matrices are non-negative, to the intuition that negative matrices for users and items are meaningless. It uses regularized stochastic gradient descent to optimize the loss function and the prediction equation is the same as SVD. I implemented it using the NMF model of the Python surprise library.

Similarly, I also fine-tuned the number of factors and the number of epochs. As the Grid Search with 5-fold cross-validation results shown in Fig. 3., we can the best number of factors is 17 and the epoch is 30.

```
# Need to fine-tune the number of factors, and epoch number
param_grid = {'n_factors': range(10, 20, 1)}

gs = GridSearchCV(NMF, param_grid, cv=kf_sp, measures=['rmse'], n_jobs=-1)
gs.fit(ratings)

print(gs.best_score['rmse'], gs.best_params['rmse'])
✓ 34.5s
1.056567451480814 {'n_factors': 17}


param_grid = {'n_epochs': range(20, 100, 10)}

gs = GridSearchCV(NMF, param_grid, cv=kf_sp, measures=['rmse'], n_jobs=-1)
gs.fit(ratings)

print(gs.best_score['rmse'], gs.best_params['rmse'])
✓ 27.1s
1.0567111132222702 {'n_epochs': 30}
```

Figure 3: GridSearchCV for hyperparameters of NMF

2.5 Neural Network

Apart from matrix factorization, the neural network is also very popular for building a recommendation system. The basic idea of the neural network is first embedding the input users/items id and concatenating them together, and then going through several fully connected layers to allow the model to learn the relationship between users and items. I implemented the network using Python TensorFlow.

Neural networks are very difficult to fine-tune the hyperparameters. I observed that a larger number of factors/nodes can result in higher accuracy but will also increase the overfitting problem. The comparison of different number of factors is shown in Fig. 4. and we can see 32 works best. A dropout layer is added to control the overfitting problem. Thus, I decided to embed both users and movies to 32 nodes and then go through a dense layer with 16 nodes, a dropout layer, and another dense layer to output and then go through a lambda function to have a suitable range to compare with ratings. The final structure is shown in Fig. 5.

```
n_factors=16    mean RMSE=0.9627    with std=0.0245
n_factors=32    mean RMSE=0.9625    with std=0.0242
n_factors=50    mean RMSE=0.9648    with std=0.0262
n_factors=64    mean RMSE=0.9632    with std=0.0238
```

Figure 4: Neural network hyperparameter tuning

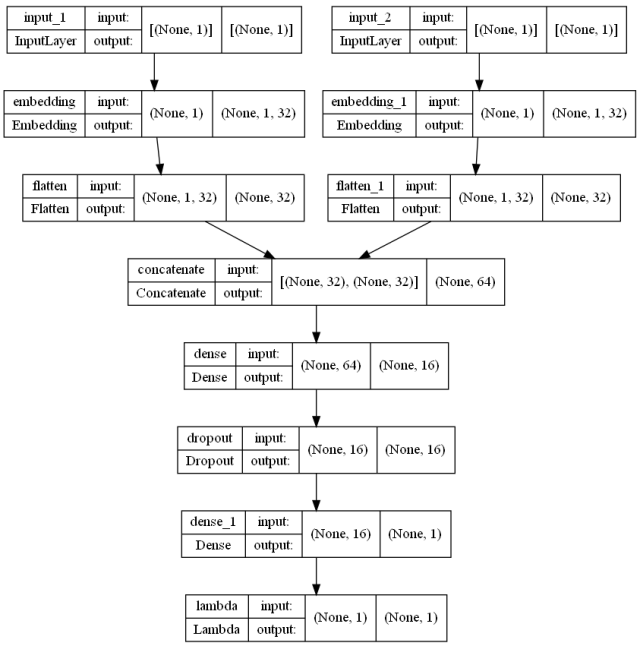

Figure 5: Structure of neural network

3 Data

I used the latest MovieLens 100K dataset (2018 version). The ratings.csv contains 100,836 ratings and tags.csv contains 3683 tags. The ratings.csv has 610 unique users and 9724 unique movies. The tags.csv has 58 unique users and 1572 unique movies. We can see the number of tags is much less than ratings, which may cause much null value when predicting using KNN.

4 Results

To better compare the performance, I perform a 5-fold cross-validation and make sure for each turn, each algorithm always uses the same partition of the dataset. Then, I measured the performance in three-dimension: Root of Mean Square Error

(RMSE), Mean Absolute Error (MAE), and Predicting Time (Time), as shown in Table 1.

	RMSE	MAE	Time
Random	1.4263	1.1387	0.1021
NMF	1.0429	0.8296	0.0902
SVD	0.9640	**0.7480**	**0.0746**
KNN	1.4478	1.2480	35.6309
Neural Network	**0.9627**	0.7525	0.6441

Table 1: Performance Table

	Random	NMF	SVD	KNN	Neural Network
Random	0	0.96	1.06	0.97	1.05
NMF	0.96	0	0.44	**0.12**	0.43
SVD	1.06	0.44	0	0.45	**0.23**
KNN	0.97	0.12	0.45	0	0.44
Neural Network	1.05	0.43	0.23	0.44	0

Table 2: Consistency Table using RSME

Table 1 shows the results compared with the ground truth and three metrics are all smaller-is-better. We can see SVD results in best MAE and short prediction time while neural network has best RMSE. We also can see the worst two are random and KNN. This is consistent with my expectation because random does not learn from the training dataset but instead randomly guesses using a normal distribution. Similarly, although KNN works well when tags are available, there are only 3000+ tags compared to 100,000+ ratings. Thus, for most data, KNN is also randomly guessing, which results in poor scores. KNN also needs to calculate the distance between all pairs of data points, which results in high time consumption. Besides, we also can see that the performance of NMF is close to SVD, which makes sense due to the high similarity between the NMF algorithm and the SVD algorithm. Neural network works well in RMSE and MAE but requires longer prediction time. Overall, we can see that NMF, SVD, and neural networks has much lower RMSE and MAE than random, which means they are much superior to randomly guessing.

Table 2 shows the consistency between results from each algorithm, measured by RMSE. For this part, I split the dataset into 75%, 25% and predict the testing part to compare. We can see that the most similar pair is NMF and KNN. This may be caused by the coincidence that their predictions are very similar between NMF and KNN. The second likely pair is SVD and neural network. This makes sense because they both perform well and may both be close to the ground truth values. Overall, we can see the other four algorithms share much lower RMSE (at least half of error) than pairs with random guessing, which means they are more similar with each other than with random.

5 Conclusion

In conclusion, I believe my five algorithms perform as expected. Four algorithms except random guessing actually learn patterns and can provide valuable recommendations. SVD performs best and the neural network also works fine. NMF needs further hyperparameters tunings, such as the initial values, to improve the performance. KNN needs larger datasets of tags to provide more information and reduce the null values.

6 Future Work

As we can see, there are still much space to improve these algorithms. For KNN, we may want to introduce more movie tags data and may try more distance calculation methods (e.g. cosine distance). For neural networks, if we can have a larger dataset, we can increase the number of nodes in each layer. For NMF, which is very sensitive to the initial value, we can try to use SVD as the initializer for NMF. We also want to increase the regularization term to control the overfitting.