Recommender System for Small MovieLens dataset

Comparison of Performance Between 5 Different Algorithms

Nhien Phan
Department of Computer and Data Sciences
Case Western Reserve University
Cleveland, OH, USA
nhien.phan@case.edu

ABSTRACT

The "movielens-100k" dataset is a classic benchmark dataset for recommender systems research, and it contains user ratings of movies on a scale of 1 to 5. It is often used as a benchmark dataset for recommender system research because it is small enough to be easily analyzed and experimented with, yet large enough to provide meaningful results. It has been widely used in research studies and competitions, and many recommendation algorithms have been developed and tested on this dataset.

I use 5 different algorithms to build a recommender system for this dataset. These include random, k-nearest neighbors, SVD, matrix factorization with regularization, and deep neural network. The algorithm with the best performance in terms of RMSE is X while the one with the best performance assessed by MAE is Y. The algorithm that takes the lowest runtime to produce the recommendations is Z with the runtime of 100.

ALGORITHMS, RMSE, MAE AND RUNNING TIME

For all models, I split the "movielens-100k" data set into training data set and testing data set. I use GridSearchCV for hyperparameter tuning for the random, SVD, and matrix factorization with a regularization algorithms; using RMSE as the criteria. For the deep neural network algorithm, I use the keras-tuner component of Keras to perform hyperparameter tuning. Five-folds cross validation is also performed for each algorithm.

1. Random:

For the random, I import the NormalPredictor module from the Surprise library. This is the most basic algorithm for the construction of a recommender system, which predicts the rating of each item randomly. Thus, there is no hyperparameter tuning involved with this algorithm.

- RMSE: 1.52.
- MAE: 1.22.
- Running time: 0.1575 seconds.

2. K-Nearest Neighbors (kNN):

For the k-nearest neighbors algorithm, I import the KNNBasic module from the Surprise library. I tune the k hyperparameter (number of nearest neighbors) ranging from 40 to 200. Using RMSE as the criteria, I found k = 40 resulted in the best performance for the algorithm.
- RMSE: 0.98.
- MAE: 0.78.
- Running time: 3.7444 seconds.

3. Singular Value Decomposition (SVD):

For the SVD algorithm, I import the SVD module from the Surprise library. The hyperparameter that required tuning for this algorithm is “n_factors” (number of factors), “n_epochs” (number of iterations of the SGD procedure), “lr_all” (learning rate for all parameters), and “reg_all” (regularization term for all parameters). The result from GridSearchCV tuning showed that n_factors = 50, n_epochs = 25, lr_all = 0.006, reg_all = 0.01 gives the best performance in terms of RMSE.
- RMSE: 0.93.
- MAE: 0.73.
- Running time: 1.05 seconds.

4. Matrix factorization with regularization:

For the matrix factorization with regularization model, I decide to create my own implementation of the algorithm instead of using the provided implementation after discussion with the TA to improve runtime performance. The stochastic gradient descent (SGD) operation in my implementation uses the "fit" method to update the biases and latent factor matrices for each rating in the training set, and then compute the RMSE on the training set for each epoch. The "test" method computes the predicted rating for a given user-item pair by summing the global bias, user bias, item bias, and the dot product of the user and item latent factor matrices. The hyperparameter of my implementation is similar to that of the SVD algorithm, which includes the following hyperparameters: number of factors (n_factors), the number of epochs (n_epochs), the learning rate (lr), and the regularization term (reg). This is because SVD is a subcategory of matrix factorization, so I based my hyperparameters on SVD's hyperparameters.

The hyperparameter tuning step uses GridSearchCV once again, which results in n_factors = 200, n_epochs = 30, lr_all = 0.01, reg_all = 0.1 giving the best performance.
- RMSE: 1.51.
- MAE: 1.20.
- Running time: 180.71 seconds.

5. Deep Neural Network

For the deep neural network algorithm, I import the Tensorflow and the Keras library. I also import the Keras Sequential model to build the deep neural network. The hyperparameter tuning step for this algorithm is performed with RandomSearch module in the Keras-Tuner library – a library specifically used for hyperparameter tuning involving the Keras library. I combined the RandomSearch of Keras-Tuner with my own implementation to find the optimal hyperparameters.

I create a "create_model" function which takes a hyperparameter object "hp" as input and returns a Keras neural network model with the specified hyperparameters. I use the "hp.Int" method to define integer-valued hyperparameters "units1" and "units2", which represent the number of units in the first and second hidden layers respectively. Regarding the dropout rates in the first and second dropout layers, I use the "hp.Float" method to define float-valued hyperparameters "dropout1" and "dropout2". Then, I use the "hp.Choice" method to define a categorical hyperparameter "learning_rate", which represents the learning rate of the Adam optimizer. Each of these inner methods of "create_model" will go over a range of values, defined by the boundary of a "min_value" and a "max_value" to determine the best hyperparameters.

The hyperparameter tuning step give the result that units1 = 512, dropout1 = 0.1, units2 = 128, dropout2 = 0.3, learning_rate = 0.01 gives the best performance.
- RMSE: 1.11.
- MAE: 0.94.
- Running time: 13.35 seconds.

CROSS-ALGORITHMS RMSE BY PAIRS

Below is the table that shows the RMSE between the recommendations produced by each algorithm (pair-wise calculation for comparison). This measure shows the degree of similarity between the predictions produced by one model to another.

	Random	kNN	SVD	Matrix Factorization	Deep Neural Network
Random	**0**	1.52	0.93	1.43	1.03
kNN	1.52	**0**	0.93	1.15	0.58
SVD	0.93	0.93	**0**	1.2	0.68
Matrix Factorization	1.43	1.15	1.2	**0**	1.04
Deep Neural Network	1.03	0.58	0.68	1.04	**0**

Table 1: Cross-model RMSE for each pair of models.

The table shows that the difference of recommendations produced by Random and SVD is similar to that of the kNN-SVD pair – both have the RMSE of 0.93. The two algorithms that produced the most similar recommendations are between Deep Neural Network and kNN (RMSE = 0.58), followed closely by Deep Neural Network and SVD (RMSE = 0.68). Meanwhile, on the other end of the spectrum, Random and kNN algorithms produced the most different recommendations from each other with the RMSE of 1.52.

CONCLUSION

Regarding the five algorithms, we can see from the result above that SVD is the best performer in terms of both RMSE and MAE, with its running time being the second best. The kNN algorithm is the second best in RMSE and MAE by a small margin of difference compared to SVD. The random algorithm has the best running time performance but its RMSE and MAE are the highest among five algorithms. This means it has the lowest accuracy when making recommendations. The matrix factorization algorithm is equal with random algorithm in RMSE and MAE performance with neglient difference. However, the runtime of the matrix factorization algorithm is the longest – approximately 3 minutes – while other algorithms only take seconds or fractions of seconds to finish. The deep neural network algorithm falls in the average range of performance in all three categories – RMSE, MAE, runtime. From this observation, it is reasonable to conclude that SVD is the algorithm with the best overall performance.
.

Personal Effort: Nhien Phan – 100%.

DISCUSSION/FUTURE WORK

Recommender systems have become an essential part of modern Internet economy, and they are used to make personalized recommendations to users. With the growth of e-commerce and the availability of data, recommender system research is expected to continue to be a hot topic in the coming years. Here are some potential directions for future work in recommender system research:

- Random Algorithm: The random algorithm is a simple baseline algorithm that randomly recommends items to users. While this algorithm is not particularly effective, it can serve as a benchmark against which more sophisticated algorithms can be compared. Future work in this area could focus on developing more advanced simple and easy to implement baseline algorithms.

- k-Nearest Neighbors (kNN): The kNN algorithm is a memory-based algorithm that uses the similarity between users or items to make recommendations. Future work in this area could focus on developing more sophisticated similarity metrics and weighting schemes, as well as on exploring the use of hybrid methods that combine memory-based and model-based approaches.

- Singular Value Decomposition (SVD): SVD is a matrix factorization technique that is commonly used in recommender systems. Future work in this area could focus on developing more advanced matrix factorization methods that incorporate additional sources of information, such as social network data or text data, as well as on exploring the use of non-negative matrix factorization techniques.

- Matrix Factorization with Regularization: Regularization is a technique used to prevent overfitting in machine learning models, and it is commonly used in matrix factorization algorithms for recommender systems. Future work in this area could focus on developing more sophisticated regularization techniques that are tailored to the specific needs of recommender systems, as well as on exploring the use of deep learning methods that incorporate regularization.

- Deep Neural Networks: Deep neural networks have been shown to be effective in a wide range of machine learning tasks, including recommender systems. Future work in this area could focus on developing more advanced deep learning architectures that are tailored to the specific needs of recommender systems, as well as on exploring the use of reinforcement learning techniques to optimize recommendations over time.

Generally, the direction for future research of recommender system is likely to involve a combination of these approaches, as well as the integration of additional sources of data (image data and user feedback data for example). As the Internet economy continues to grow at an exponential pace, it is reasonable to expect recommender system research to make significant real-world impacts in the future.

REFERENCES

[1] Random Library
Basic algorithms — Surprise 1 documentation
[2] K-Nearest Neighbor Library
k-NN inspired algorithms — Surprise 1 documentation
[3] Single Value Decomposition Library
Matrix Factorization-based algorithms — Surprise 1 documentation
[4] Deep Neural Network Library
The Sequential class (keras.io)
The Sequential model (keras.io)