Comparing Algorithms for Recommendation Systems

CSDS 435 Project 3, Spring 2023

Group 1

John Mays
jkm100@case.edu

Aaron Orenstein
aao62@case.edu

ABSTRACT

In this report, we implement and survey five separate recommender systems (Random, K-Nearest Neighbors, SVD, Matrix Factorization, and a Deep Neural Network) on a small dataset (MovieLens-100K). We present implementation details, find optimal hyperparameters via a hyper-parameter tuning phase, and present a comprehensive comparison of the five algorithms on our dataset using multiple standard metrics.

1 Background: Algorithms

1.1 Random

The random algorithm is the most straightforward algorithm. It generates a normal distribution based on the training data. When predicting for an unseen example, it generates a totally random rating by sampling from the distribution it created.[1]

1.2 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a clustering algorithm, which may not seem like an intuitive choice for a recommender system, but can actually work very well. A new instance is given a rating based off of a weighted (by similarity to neighbors) sum of its neighbors ratings over all of the features.[1] In other words, more similar ratings influence the new rating more.

1.3 SVD

Singular value decomposition is a process of splitting a matrix into a product of two orthogonal and one diagonal matrix. In this case, we are decomposing the matrix of user movie rankings. This produces a low-dimensional representation of the rankings which is used to produce new recommendations.

1.4 Matrix Factorization

Matrix factorization is very similar to SVD except the ranking matrix is decomposed into two matrices instead of three. These matrices have none of the special properties of SVD. Instead, we can change the dimensions of these matrices to control the size of the low-dimensional feature space encoding. We use this encoding to produce new rankings as recommendations.

1.5 Deep Neural Network

Our deep neural network is structured as a multi-layer perceptron. The input data is fed directly into two appropriately-sized embedding layers that will learn how to transform categorical data (943 users and 1682 movies) into a 50-dimensional continuous representation.[3] The embedded representations are then concatenated and fed through one 10-node (fully-connected) dense layer with ReLU activation, which finally connects to a single node which is normalized and should eventually be learned as the output node.

2 Methods: Hyperparameter Tuning and Comparison

1.1 Methodology

We selected a range of values for the most relevant hyperparameters, and looked for the best performance under each set.

2.2 Algorithm-Specific Approaches

2.2.1 Random

The random approach does not require hyperparameter tuning and serves only as a benchmark that our algorithm should outperform.

2.2.2 K-Nearest Neighbors

For KNN, we decided only to test *k,* the number of neighbors. The results for this are summarized in Table 3 of the Appendix. *k=20* achieved the best performance

2.2.3 Support Vector Decomposition

For SVD, we tested the number of factors and epochs as well as the learning rate and regularization factor. epochs and learning rate are tradeoffs of convergence speed and accuracy. Factors and regularization purefly affect accuracy. Results are shown in Table 2.

2.2.4 Matrix Factorization

Matrix factorization takes two parameters: number of latent features (K) and number of steps. The former must be tailored to the specific use case and the latter is a general way to balance speed and accuracy. MF takes a long time to run but we found that increasing both these values improves results (see Table 4).

2.2.5 Deep Neural Network

There are so many hyperparameters you can tune with a neural network, so we decided to let state of the art/norms dictate some of them so we could narrow in on a few: hidden layer size, hidden layer depth, and learning rate.So, very little time is saved for each method, and the performance metrics do not differ massively either. By both RMSE and MAE, the best algorithm was a (10→1) dense layer structure, while the worst was no dense layers at all (just dot). See table 5.

With this structure in mind, we also tested several learning rates to see if we could improve the performance:

There is a certain range of learning rates (approx. [5e-4,1e-2]) that makes negligible differences in performance, but exceeding that range begins to significantly increase error as optima can no longer be reliably found/sat in.

3. Results: Overall Comparison of Tuned Algorithms

3.1 Performance

MF ran the slowest by far while Random was the fastest. DNN, SVD, and KNN performed adequately (listed here in order of ascending runtime). The order of increasing performance is: Random, MF, KNN, SVD, DNN.

3.2 Similarity

To compare algorithms, we evaluated the results of one algorithm using another algorithm's results as the ground truth. Of this, Random was clearly very different from all algorithms. KNN had the three highest degrees of similarity with the remaining algorithms. DNN, SVD, and MF were all equally but less similar to each other.

4 Discussion

4.1 Future Work

We recognize that the optimization and comparison done here is neither complete nor comprehensive. To reach more certain conclusions about our algorithms, more metrics (like std. across folds), more complete searches through tuning spaces, and more tuning parameters would have to be done. We are sure that some of these changes could undoubtedly augment the results and conclusions of our paper.

4.2 Conclusion

Overall, DNN and SVD had the best balance of speed and accuracy. KNN is unilaterally less performant than SVD. MF has too high of a time cost to be viable, even though it could potentially outperform SVD and DNN with more iterations. All algorithms outperformed Random, as expected.

5. Statement of Participation

We contributed equally (50% each) to this project. All team members agree with the specified effort.

References

[1] Surprise Documentation. Online. Last accessed 4

[2] Wittenauer, John. *Deep Learning With Keras: Recommender System.* Online. Last accessed 30 April, 2023. URL: https://medium.com/@jdwittenauer/deep-learning

[3] Yeung, Albert A. *Matrix Factorization: A Simple Tutorial and Implementation in Python.* Online. Last accessed 30 April, 2023. URL: http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/#implementation-in-python

Appendix

Our results have been included here. All times are in seconds. Hyperparameter tuning results are averages over all 5 fold for each parameter set.

Table 1: Results for each algorithm

SVD	Fold0	Fold1	Fold2	Fold3	Fold4	Avg
RMSE	0.937	0.931	0.929	0.934	0.943	0.935
MAE	0.741	0.733	0.732	0.735	0.743	0.737
Time	1.388	1.593	1.564	1.628	1.397	1.514

KNN	Fold0	Fold1	Fold2	Fold3	Fold4	Avg
RMSE	0.975	0.969	0.975	0.976	0.982	0.975
MAE	0.769	0.764	0.769	0.769	0.776	0.769
Time	3.586	3.2	3.139	4.55	4.879	3.871

Rand	Fold0	Fold1	Fold2	Fold3	Fold4	Avg
RMSE	1.521	1.526	1.514	1.526	1.521	1.522
MAE	1.225	1.227	1.216	1.226	1.221	1.223
Time	0.255	0.312	1.098	0.302	0.291	0.452

MF	Fold0	Fold1	Fold2	Fold3	Fold4	Avg
RMSE	0.991	0.983	0.985	0.99	0.996	0.989
MAE	0.786	0.78	0.783	0.784	0.788	0.784
Time	103.3	103.5	96.90	93.01	138.5	107.0

DNN	Fold0	Fold1	Fold2	Fold3	Fold4	Avg
RMSE	0.93	0.927	0.932	0.926	0.944	0.932
MAE	0.731	0.727	0.733	0.728	0.741	0.732
Time	1.102	1.344	0.941	1.367	1.114	1.174

Table 2: Hyperparameter tuning for SVD

n_factors

	10	20	50	100
RMSE	0.938	0.934	0.933	0.935
MAE	0.74	0.737	0.736	0.737
Time	0.615	0.682	0.884	1.313

n_epochs

	10	20	50
RMSE	0.946	0.935	0.964
MAE	0.749	0.737	0.755
Time	0.717	1.219	2.69

lr_all

	0.0025	0.005	0.01	0.025
RMSE	0.947	0.935	0.952	0.981
MAE	0.749	0.737	0.747	0.769
Time	1.345	1.271	1.301	1.218

reg_all

	0.01	0.025	0.05	0.1
RMSE	0.941	0.933	0.93	0.937
MAE	0.741	0.735	0.735	0.743
Time	1.418	1.428	1.417	1.373

Table 3: Hyperparameter tuning for KNN

k	10	20	30	40	50	100
RMSE	0.988	0.975	0.976	0.978	0.98	0.992
MAE	0.777	0.769	0.77	0.772	0.775	0.786
Time	3.111	3.172	4.04	5.612	4.105	3.356

Table 4: Hyperparameter tuning for MF

K, steps

	5,5	5,10	10,5	10,10	10,25
RMSE	1.471	1.193	1.135	1.047	0.989
MAE	1.208	0.953	0.917	0.836	0.784
Time	11.828	24.997	18.463	45.434	107.028

Table 5: Hyperparameter tuning for DNN

hidden layers

	100,10,1	10,10,1	10,1	None(Dot)
RMSE	0.948	0.938	0.935	0.970
MAE	0.743	0.738	0.736	0.763
Time	0.84	0.78	0.78	0.75

learning rate

	0.0005	0.001	0.005	0.01	0.05
RMSE	0.933	0.935	0.933	0.937	1.085
MAE	0.734	0.736	0.734	0.739	0.830

Time	0.87	0.78	0.75	0.84	0.80

Table 6: Relative RMSE and MAE results

RMSE	SVD	KNN	Rand	MF	DNN
SVD		0.412	1.206	0.379	0.343
KNN	0.412		1.18	0.471	0.441
Rand	1.206	1.18		1.209	1.254
MF	0.379	0.471	1.209		0.456
DNN	0.343	0.441	1.254	0.456	

MAE	SVD	KNN	Rand	MF	DNN
SVD		0.307	0.976	0.292	0.265
KNN	0.307		0.956	0.353	0.329
Rand	0.976	0.956		0.979	1.012
MF	0.292	0.353	0.979		0.349
DNN	0.265	0.329	1.012	0.349	