Assignment 1: Recommender Systems

Group 28

Shuang Fan sxxxxxxxx

Kaiteng Jiang sxxxxxxxx

Shupei Li s3430863

1 Recommender systems

1.1 Naive Approaches

1.2 UV Matrix Decomposition

1.3 Matrix Factorization

1.3.1 Experimental Set-up

Matrix factorization algorithm in `gravity-Tikk.pdf` consists of three stages — initialization, gradient descent, and evaluation. In the experiments, we initialize feature matrices $\mathbf{U}_{I\times K}$ and $\mathbf{M}_{K\times J}$ from a Gaussian distribution $N \sim (0, 0.1)$, where I, J, and K are maximal UserID, maximal MovieID, and the number of features respectively. `gravity-Tikk.pdf` combines gradient descent and regularization strategies. Main update formulas are,

$$
\begin{aligned}
u_{ik}^{(t+1)} &= u_{ik}^{(t)} + \eta \cdot \left(2e_{ij} \cdot m_{kj}^{(t)} - \lambda \cdot u_{ik}^{(t)}\right) \\
m_{kj}^{(t+1)} &= m_{kj}^{(t)} + \eta \cdot \left(2e_{ij} \cdot u_{ik}^{(t)} - \lambda \cdot m_{kj}^{(t)}\right)
\end{aligned}
$$

where η, λ are hyperparameters, and t represents the t th iteration. To enhance the efficiency of program, we update the weights based on the rows or columns of matrices, that is,

$$
\begin{aligned}
U^{(t+1)}[i,:] &= U^{(t)}[i,:] + \eta\left(2e_{ij}M^{(t)}[:,j] - \lambda U^{(t)}[i,:]\right) \\
M^{(t+1)}[:,j] &= M^{(t)}[:,j] + \eta\left(2e_{ij}U^{(t)}[i,:] - \lambda M^{(t)}[:,j]\right)
\end{aligned}
$$

Termination condition is achieving the maximum iteration specified by user. In the evaluation stage, RMSE and MAE are chosen as metrics. Because U and M are real value matrices, the predicted value is a real number that may exceed the valid range of rating. To fix this problem, we truncate the predicted value according to the function,

$$
f(\hat{x}) = \begin{cases} 1, & \hat{x} < 1, \\ \hat{x}, & 1 \leq \hat{x} \leq 5, \\ 5, & \hat{x} > 5. \end{cases}
$$

We try five different sets of hyperparameters to improve the model performance. Table 1 summarizes all hyperparameters in matrix factorization algorithms. It is worth noting that we set the random seed to 1 both in weights initialization and five-fold division, which ensures reproducibility. All experiments of Task 1.3 are run on a multi-core CPU Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz. We adopt multiprocessing programming to speed up the program.

Table 1: Hyperparameters in Matrix Factorization

Hyperparameter	**Meaning**
seeds	The random seed. Default: 1.
num_factors (K)	The number of features.
num_iter (N)	The maximum iteration.
regularization (λ)	Regularization rate.
learn_rate (η)	Learning rate.

1.3.2 Results

Table 2 reports RMSE, MAE, and the actual run time of matrix factorization algorithm on MovieLens 1M data, with different hyperparameter settings. RMSE and MAE are the mean values of five folds.

Table 2: Results of Task 1.3

K	N	λ	η	**Train RMSE**	**Train MAE**	**Test RMSE**	**Test MAE**	**Time**
*10	75	0.05	0.005	0.7689	0.6036	0.8686	0.6785	18m 37s
20	75	0.05	0.005	**0.7003**	**0.5475**	0.8848	0.6878	18m 38s
10	100	0.05	0.005	0.7673	0.6020	0.8670	0.6793	24m 38s
10	75	0.01	0.005	0.7627	0.5949	0.8807	0.6829	18m 29s
10	75	0.05	0.001	0.7980	0.6292	**0.8611**	**0.6763**	18m 34s

* This set of hyperparameters is suggested by Task 1.3.

According to Table 2, the suggested setting is not the optimal choice.

1.3.3 Algorithm Analysis

Notations

As stated in the question,

- M: The number of movies.
- U: The number of users.
- R: The number of ratings.

The number of features K is an integer specified by user. We regard K as a constant in the following analysis.

Time Complexity

The time complexity of initializing U and M depends on the implementation of random number generation algorithm. For simplicity, we assume the time complexity of the initialization stage is $O(1)$. According to our Python implementation, the `for` loop to update weights has the time complexity $O(R)$. In the `for` loop, calculating error, computing gradients, and updating weights all have the time complexity $O(K) \to O(1)$. To evaluate the performance, we need to traverse the rating table, which leads to $O(R)$ time complexity. Calculating RMSE and MAE also has time complexity $O(R)$. Therefore, the time complexity of our implementation is $O(R)$.

Memory Complexity
We need to initialize U, M, and rating table at the beginning of the algorithm. This step requires $O(UK + MK + R) \rightarrow O(U + M + R)$ memory. In the `for` loop, storing error value and gradients needs $O(1)$ memory. During the evaluation, storing predicted values requires $O(R)$ memory and storing metrics needs $O(1)$ memory. Therefore, the memory complexity of our implementation is $O(U + M + R)$.

1.4 Comparison of Algorithms

Table 3 compares the performance of previously mentioned algorithms, which only includes the model with the best performance on test data if multiple hyperparameters have been explored.

Table 3: Algorithm Comparison

Algorithm	**Train RMSE**	**Train MAE**	**Test RMSE**	**Test MAE**	**Time**
Matrix Factorization	0.7980	0.6292	0.8611	0.6763	18m 34s

2 Data visualization