Assignment 1: Recommender Systems

Group 28

Shuang Fan s3505847
Kaiteng Jiang s3479420
Shupei Li s3430863

1 Recommender systems

RMSE and MAE are chosen as metrics in Task 1.1-1.3. The predicted value in all task 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,

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

It is worth noting that we set the random seed to 1 in all models that involve randomness, e.g. weights initialization, five-fold division, etc. The fixed random state ensures reproducibility.

1.1 Naive Approaches

1.1.1 Experimental Setup

During the sampling process of an "average rating" recommender, some users or some movies might disappear from the training sets. To fix this problem, we need to define a fall-back value. In our experiments, we build models of global average, user average, movie average and a linear combination of user and movie averages (with and without the intercept). A linear regression model can be expressed as,

$$pred = \alpha \cdot avg_{user} + \beta \cdot avg_{movie} + \gamma,$$

where γ is the intercept. After simplifing the variant γ, we get,

$$pred = \alpha \cdot avg_{user} + \beta \cdot avg_{movie}.$$

Considering the matrix $\mathbf{U}$ with UserID and MovieID, the global average should be the average value of all existed Ratings. For user average rating, we calculate every user's average rating. This is the avg_{user} term in the equation. For movie average rating, we calculate every movie's average rating. This is the avg_{movie} term in the equation. When implementing the linear regression algorithm, we calculate the coefficient and consider the situations that are with and without the intercept γ.

All experiments of Task 1.1 are run on a multi-core CPU Intel(R) Core(TM) i9-9880H CPU @ 2.30GHz.

1.1.2 Results

Table 1 reports RMSE, MAE, and the actual run time of global average, user average, movies average and a linear combination of user and movie averages(with and without the intercept). RMSE and MAE are the mean values of five folds.

Table 1: Results of Task 1.1

Algorithm	**Train RMSE**	**Train MAE**	**Test RMSE**	**Test MAE**	**Time**
GlobalAvg	1.1171	0.9339	1.1171	0.9339	0.93s
UserAvg	1.0277	0.8227	1.0355	0.8290	1.38s
MovieAvg	0.9742	0.7783	0.9794	0.7823	1.43s
LinearReg	**0.9145**	**0.7248**	**0.9002**	**0.7122**	13m 57s
LinearRegNI	0.9465	0.7586	0.9345	0.7487	13m 45s

1.1.3 Algorithm Analysis

Notations
As stated in the question,

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

Time Complexity
The code in the `for` loop is executed n times. Therefore, its run time is proportional to n, which means its time complexity is $O(n)$. We need to fill a $M \times U$ matrix with the generated ratings, which is a matrix with movies and users. To solve this, we need a $O(MU)$ time complexity. Calculating RMSE and MAE also has time complexity $O(MU)$. Therefore, the time complexity of our implementation is $O(MU)$.

Memory Complexity
In the `for` loop, storing error value and the RMSE and AME needs $O(1)$ memory. During the *global_average*,*user_average*, *movie_average*, *linear regression with and without intercept*, the memory complexity of our implementation is $O(MU)$.

1.2 UV Matrix Decomposition

1.2.1 Experimental Set-up

The UV matrix decomposition is an element-wise update algorithm of the feature matrices $U_{m\times k}$ and $V_{k\times n}$, of which the multiplication UV approximates the utility matrix $M_{m\times n}$. The goal of update is to minimize the mean square error (MSE) function. For an element u_{rs} of the matrix U, the optimization problem can be written as,

$$u_{rs} = \min_x \sum_{j, m_{rj}\neq 0} \left[m_{rj} - \left(\sum_{k\neq s} u_{rk}v_{kj} + xv_{sj} \right) \right]^2 .$$

And for v_{rs},

$$v_{rs} = \min_{y} \sum_{i, m_{is} \neq 0} \left[m_{is} - \left(\sum_{k \neq r} u_{ik} v_{ks} + u_{ir} y \right) \right]^2 .$$

They both have closed-form solutions,

$$x = \frac{\sum\limits_{j, m_{rj} \neq 0} v_{sj} \left(m_{rj} - \sum\limits_{k \neq s} u_{rk} v_{kj} \right)}{\sum\limits_{j, m_{rj} \neq 0} v_{sj}^2},$$

$$y = \frac{\sum\limits_{i, m_{is} \neq 0} u_{ir} \left(m_{is} - \sum\limits_{k \neq s} u_{ik} v_{ks} \right)}{\sum\limits_{i, m_{is} \neq 0} u_{ir}^2}.$$

In the experiment, we visit every element of both U and V in a random order per epoch. And predicted values are clipped to the range of $[1, 5]$, as mentioned at the beginning of the section. Table 2 explains the hyperparameters in the algorithm.

Table 2: Hyperparameters in UV Matrix Factorization

Hyperparameter	**Meaning**
seeds	The random seed. Default: 1.
num_factors (K)	The number of features.
num_iter (N)	The maximum iteration.

1.2.2 Results

Table 3 reports the results of the task.

Table 3: Results of Task 1.2

K	N	**Train RMSE**	**Train MAE**	**Test RMSE**	**Test MAE**	**Time**
5	30	0.8270	0.6486	0.8979	0.7005	72m 25s

1.2.3 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
Computing the sum $\sum_k$ needs time $O(K)$, and $\sum_j$ needs $O(M)$. Thus, a single update of an element of U costs $O(MK)$. Similarly, updating an element of V needs $O(UK)$. Since there are UK and KM elements in U and V respectively, an epoch which updates all the elements of U and V has a time complexity of $O(UK \cdot MK + KM \cdot UK) = O(MUK^2) \to O(MU)$. This is a very costly and slow algorithm, which explains the long running time in the experiment result. In fact, we have tried to set the dimension K to larger numbers, like 10, and it took many hours to run a single experiment, which became unacceptable.

Memory Complexity
We only need to initialize, store and update three matrices: U, V and M, thus the memory complexity is $O(UK + MK + R) \to O(U + M + R)$.

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,

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

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,

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

Termination condition is achieving the maximum iteration specified by user.

We try five different sets of hyperparameters to improve the model performance. Table 4 summarizes all hyperparameters in matrix factorization algorithms. 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.

1.3.2 Results

Table 5 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.

According to Table 5, the suggested setting is not the optimal choice.

Table 4: 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.

Table 5: 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.

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 $O(UK + MK + R) \to O(U + M + R)$ 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 6 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 6: Algorithm Comparison

Algorithm	**Train RMSE**	**Train MAE**	**Test RMSE**	**Test MAE**	**Time**
GlobalAvg	1.1171	0.9339	1.1171	0.9339	0.93s
UserAvg	1.0277	0.8227	1.0355	0.8290	1.38s
MovieAvg	0.9742	0.7783	0.9794	0.7823	1.43s
LinearReg	0.9145	0.7248	0.9002	0.7122	13m 57s
LinearRegNI	0.9465	0.7586	0.9345	0.7487	13m 45s
UV Decomposition	0.8270	0.6486	0.8979	0.7005	72m 25s
Matrix Factorization	**0.7980**	**0.6292**	**0.8611**	**0.6763**	18m 34s

According to Table 6, Matrix Factorization algorithm outperforms all the other algorithms both on training set and test set, while its real run time is acceptable.

2 Data visualization