Advances in Data Mining
Assignment 1 - Recommender Systems

Dimitrios Ieronymakis
d.ieronymakis@umail.leidenuniv.nl
Maria Ieronymaki
m.ieronymaki@umail.leidenuniv.nl

Contents

Introduction

This report describes the steps required for the implementation of several recommendation algorithms for the MovieLens 1M dataset and discusses the results obtained by estimating their accuracy with the Root Mean Squared Error (RMSE), and the Mean Absolute Error (MAE). To make sure that the results are reliable, the 5-fold cross-validation has been used.
The three implemented algorithms are: the Naive Approaches, the UV matrix decomposition and the matrix factorization.
All the experiments were run on an Intel core i7-10700K processor clocked at 3.80Ghz paired with a DDR4 SDRAM clocked at 2400Mhz. The seed used to replicate our results was set with numpy.random.seed(0).

Dataset and observations

The MovieLens 1M dataset contains about 1.000.000 ratings given to about 4.000 movies by about 6.000 users. The data is divided in three data sets: the ratings.dat file which contains movie-user pairs with a rating and the timestamp, the users.dat and the movies.dat files that provide some information about users (gender, age, occupation, zip code) and movies (genre,title), respectively. Relationships between the different datasets are established by using two key attributes: UserID and MovieID.
Missing values have been searched in the data sets, but no NaN values are present.
Since not all movie-user pairs have a rating, in order to avoid complications, for instance when sampling and evaluating our models, the global average rating has been used as a fall-back value in all three approaches. Moreover, in order to have valid ratings, the predicted values have been rounded to 5 if they are higher than 5 and to 1 if lower than 1.

Naive Approaches

The Naive Approaches implement the 5 formulas for the the global average rating, the average rating per movie, the average rating per user, and an "optimal" linear combination of the two averages (per user and per movie), with and without the parameter γ representing the intercept value.

Global average rating

The global average rating is the simplest recommender. The algorithm uses the kFold (k=5) method to split the data into 5 parts of more or less equal sizes. Five 'models' are built and each of them is applied to the training and test set. The predictions of each model correspond to the global average of the ratings calculated on the specific train split data. To proceed with the estimation of the accuracy, the root mean square error and the mean absolute error between the predictions on the train and test set and the actual ratings have been estimated. The results generated are 5 different estimates of the accuracy; their average is considered to be a good estimate of the error on the future data.

Results are displayed in Table 1 below:

Error	Train set	Test set
RMSE	1.117100	1.117103
MAE	0.93385	0.93386

Table 1: Mean Square Error and Mean Absolute Error on the train and test sets of the global average rating model.

The run time for this experiment was around 0.4 seconds.

Average rating per movie

For the average rating per movie model we again implemented the 5 fold cross-validation. This time the average is calculated for each MovieID present in the split. An important thing to take into account is the fact that when splitting the data, because of its randomness, some movies or users might disappear from the training sets and appear in our test sets during the evaluation step. To deal with this issue, we found the gaps by left-merging the test data with the training data on the MovieID property and filled them with the global average rating calculated as the mean value of the current's train data ratings. The next step was the determination of the model's performance by calculating the root mean square error and the mean absolute error between the predictions and the true values for each split. Again, the mean of the five accuracies is considered to be a good estimate of the error of the model.

The results are reported in Table 2:

Error	Train set	Test set
RMSE	0.9742	0.9794
MAE	0.7783	0.7823

Table 2: Mean Square Error and Mean Absolute Error on the train and test sets of the average rating per movie model.

The run time of the second naive approach experiment was around 0.8 seconds.

Average rating per user

The model based on the average rating per user has been built following the same steps of the average rating per movie model. The only difference is that the average has been calculated for each UserID present in the split. The experiment, similarly to the one above took 0.8 seconds to run.

The results are shown in Table 3:

Error	Train set	Test set
RMSE	1.0276	1.0355
MAE	0.8227	0.8289

Table 3: Mean Square Error and Mean Absolute Error on the train and test sets of the average rating per user model.

Linear Regression

For the linear regression model we want to approximate the 'Ratings' column, as a linear combination of 2 columns: Average Rating per user (R_user), Average Rating per movie (R_movie) and a constant term γ if we include the intercept parameter. The formula for the linear regression is as follows:

$$\text{pred} = \alpha \cdot \text{R\_user} + \beta \cdot \text{R\_movie (without intercept)}$$

$$\text{pred} = \alpha \cdot \text{R\_user} + \beta \cdot \text{R\_movie} + \gamma \text{ (including intercept)}$$

α, β and γ are parameters that minimize the sum of the squared errors and are determined by using the scikit-learn package for linear regression: sklearn.linear model.LinearRegression. In this case, we calculated the average rating per movie and per user on each train split and used these parameters to train the linear regression model. The model is then evaluated on the train and the test set. The mean square error and the mean absolute error shown in Table 4 and Table 5 correspond to the mean of the 5 errors estimated in each split between the predictions and the real ratings provided on both train and test set.

Error	Train set	Test set
RMSE	0.9465	0.9344
MAE	0.7585	0.7486

Table 4: Mean Square Error and Mean Absolute Error on the train and test sets of the Linear Regression model without the intercept parameter.

Error	Train set	Test set
RMSE	0.9146	0.9002
MAE	0.7250	0.7128

Table 5: Mean Square Error and Mean Absolute Error on the train and test sets of the Linear Regression model with the intercept parameter.

Running the Linear Regression experiments took around 1.7 seconds each, for a total of 3.4 seconds.

UV matrix decomposition

The UV matrix decomposition algorithm is based on Section 9.4 of the MMDS textbook[1]. The main idea is founded on the fact that users will respond and most probably rate only the movies with certain features depending on their preferences. As a consequence, the utility matrix M with n rows and m columns (n users and m movies), containing all the ratings associated to the UserID for each MovieID, is going to be sparse matrix with many NaN values. However, there is a way to estimate these blank entries by finding a matrix U with n rows and d columns and a matrix V with d rows and m columns, such that the product between them will approximate M.

[1] http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

Preprocessing & Initialization

Many approaches have been tested for the preprocessing and initialization steps, nonetheless, for simplicity, only the best one is reported. The utility matrix M has been normalized by subtracting from each element in the position m_{ij} (i row, j column), half the average of the values in row i and half the average of the values in column j, ignoring the NaN entries. The initial weights of the U and V matrices have been set to 0 due to the normalization. However, noise has also been added to each weight by adding a random normally distributed value with mean 0 and standard deviation 1.

Note: it is important not to forget, when making the predictions, to undo the normalization.

Update rules and implementation

After having initialized the weights of U and V, the next step is to adjust them repeatedly until the optimal weights, that minimize the Root Mean Square Error of the UV matrix, are found. Specifically, the update of the elements x and y for the U and V matrix respectively, lies within the calculation of the Root Mean Square Error (between M and the dot product of U and V) and setting the derivative of it equal to 0.
The analogous formulas for the optimum value that summarize the process mentioned above are:

$$x_{rs} = \frac{\sum_j v_{sj}(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj}}{\sum_j v_{sj}^2} \qquad y_{rs} = \frac{\sum_i u_{ir}(m_{is} - \sum_{k \neq r} u_{ik} v_{ks}}{\sum_i u_{ir}^2}$$

were r is the row, s is the column, $\sum_{k \neq s,r}$ is the sum for $k = 1, ..., d$ except for $k \neq s, r$ and $\sum_{i,j}$ is the sum over all i,j such that the element $m_{is,rj}$ of the matrix M is not a NaN entry.

The order in which we update the weights of U and V is row-by-row, following the round-robin method: the algorithm starts with updating the weights of each row (n) and column (d) of the U matrix and then follows the same process for the V matrix. This procedure is repeated for i iterations.

Experiments

Experimenting with the UV Matrix Decomposition method has been pretty straight forward since the only parameters that needs tuning are the parameter d (which represents the dimension of columns of U and rows of V) and the number of iterations i to repeat the update process. For the parameter d we have considered the following sizes: 2, 4, 6, 8, 16, 20, 24, 32, while for the number of iterations we have used the value of 20, which has been more than enough in most cases to reach the minimum RMSE and MAE values on the test set. The results of this process can be visualized in Table 6. As usual the resulting values are the mean of the 5-fold cross validation results:

Dimension	Train RMSE	Test RMSE	Train MAE	Test MAE	Required iterations
2	0.8669	0.8950	0.6820	0.7026	17.2
4	0.8236	0.8807	0.6465	0.6869	20.0
6	0.7927	0.8823	0.6211	0.6840	20.0
8	0.7699	0.8952	0.6022	0.6899	20.0
16	0.7009	0.9613	0.5425	0.7314	9.2
20	0.6730	0.9894	0.5175	0.7512	7.8
24	0.6484	1.0173	0.4954	0.7712	7.0
32	0.6069	1.0731	0.4579	0.8097	7.0

Table 6: Results of the experiments carried out for the UV decomposistion approach.

From Table 6 above, we can distinguish the best values of d. The ones that are able to achieve the lowest scores are d=4 and d=6, with minimal differences in the RMSE and MAE values. Also, it is important to note how higher dimensions of d have not not necessarily yield a better RMSE and MAE values. Although it takes less and less iterations for the algorithm to reach the lowest RMSE and MAE values as d increases, we can clearly see that these new lows are not as good as the ones for lower dimensions. This behaviour however, could be a consequence of the high noise value that has been used to initialize the weights. Probably if we decreased the noise for high dimensional U and V matrixes their RMSE and MAE values would be closer to the ones achieved by lower values of d.
Finally, it is worth mentioning that since the lowest errors for d= 4,6 and 8 are found in the last iteration, we could have increased the number of iterations further in order to see if lower values of the RMSE and the MAE are achievable.
A graphical representation of the behaviour of the RMSE and the MAE for d=4 can be seen in Figure 1 below.

Computation for the experiments on the UV decomposition matrix mentioned above were pretty heavy and took around 460 minutes to run (almost 8 hours).

Matrix factorization

The Matrix Factorization approach is based on the approach described in the paper "On the Gravity Recommendation System"[2]. This approach is very similar to the UV decomposition method described in the above section. The method still relies on approximating the matrix 'X' (the sparse matrix of reviews) with dimensions I x J, with the product of two matrices, 'U' of dimensions I x K and 'M' of dimensions K x J. I represents the number of users in our X matrix and J the number of movies, while K is a dimensionality hyperparameter that we need to tune.

The main difference from the UV decomposition algorithm resides on the method used to update the weights. In this case, we iterate through each non-Nan entry in our train X matrix and update the rows and columns of U and M that are responsible for predicting that specific entry. To update the weights the gradient descend approach has been used, combined with two new important parameters that control the learning of our model. Such parameters are the regularization λ parameter, that prevents our weights from reaching high values and the learning rate η parameter used to partly increase the values of the weights at each iteration.

[2] https://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/gravity-Tikk.pdf

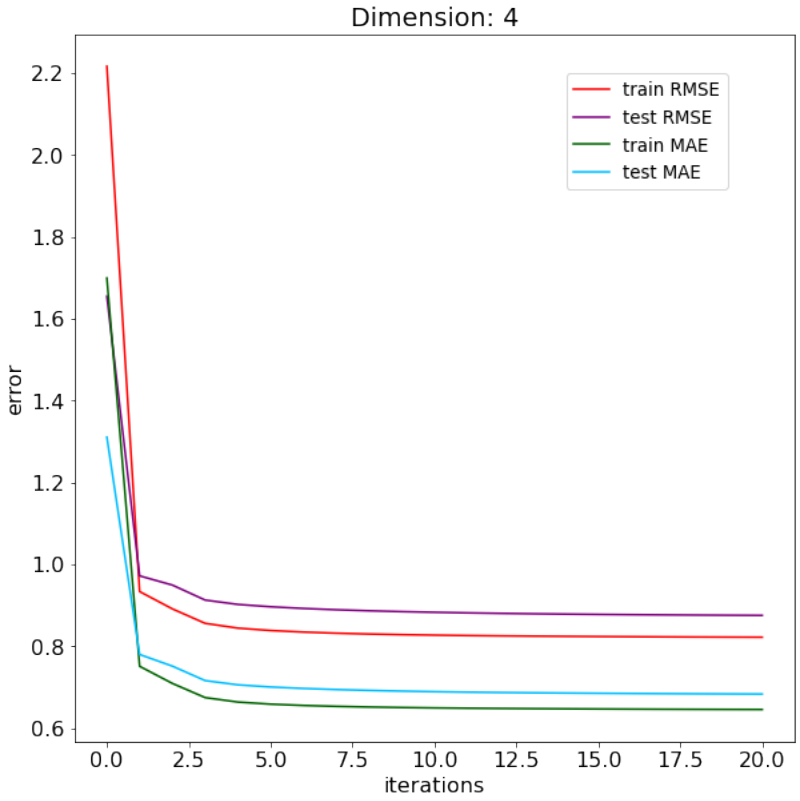

Figure 1: Graphic representation of the lowest errors on the train and test set for d= 4.

Preprocessing & Initialization

In this approach we have discovered that the initialization parameters utilized in the Matrix Decomposition algorithm described in the previous chapter are not ideal for this approach. Instead, we have experimented with the initialization parameters present in page 28, chapter 4 of the paper, which seem to work better. More specifically, we have initialized the values of U and M to 0 and applied a random normal noise of mean 0 and standard deviation of 0.01. Also, no normalization has been applied to the X matrix. As far as the hyperparameters are concerned, we have mainly used a regularization rate of 0.05, a learning rate of 0.005 while varying K in order to find the best value.

Update rules and implementation

In order to improve our estimations of the matrix X, we need to be able to update the weights of the U and M matrices. As already stated, this method relies on gradient descend with the regularization and learning rate parameters. The update rules used to update U and M are the following:

$$u'_{ik} = u_{ik} + \eta \cdot (2e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik})$$
$$m'_{kj} = m_{kj} + \eta \cdot (2e_{ij} \cdot u_{ik} - \lambda \cdot m_{ij})$$

where u_{ik} represents i-th row of the matrix U, m_{kj} represents the column j of the matrix M and e_{ij} is the error between our predicted value and the (i,j)-th non-blank training entry of X.

In order to apply these update rules, we loop through each non-blank value of X and for each one calculate the corresponding u'_{ik} and m'_{kj} that will then substitute the old values in the i-th row of U and j-th column of M. We repeat the process for i iterations and stop the algorithm when the test RMSE is not improving any more.

Experiments

In order to test the algorithm a lot of experiments have been carried out. As mentioned in the initialization paragraph, the learning rate was mainly set to 0.005 and the regularization parameter to 0.05. We have tested our algorithm for various iterations (30-60) and many sizes of K, in order to determine its best value. Specifically, the dimensions of K taken into account are the following: 8, 10, 16, 24, 30, 36, 40, 50. The results of our experiments have been summarized in Table 7. As usual the resulting values are the mean of the 5-fold cross validation results:

Dimension	Train RMSE	Test RMSE	Train MAE	Test MAE	Required iterations
8	0.7920	0.8675	0.6244	0.6800	32.4
10	0.7741	0.8648	0.6101	0.6775	27.4
16	0.7309	0.8620	0.5758	0.6759	22.4
24	0.6839	0.8604	0.5380	0.6748	21.2
30	0.6984	0.8596	0.5523	0.6738	21.0
36	0.6792	0.8589	0.5375	0.6736	20.8
40	0.6677	0.8589	0.5285	0.6736	20.6
50	0.6414	0.8577	0.5078	0.6725	20.2

Table 7: Results of the experiments carried out for the Matrix Factorization approach.

From Table 7 we can see that the lowest RMSE and MAE scores are achieved by the highest dimension: 50. It is also interesting to note that for the same values of learning rate and regularization, the more we increase the dimension of K, the less iterations of the matrix factorization algorithm are needed to reach the local minimum on the test set.
A graphical representation of the behaviour of the RMSE and the MEA values for K=50 are shown in Figure 2 below.

The final experiment we have run for the Matrix Factorization approach consists in changing the values of η and λ to try to achieve a better RMSE and MAE for the best K value calculated previously (K=50). In this experiment we have set η=0.001 λ=0.01 and the iterations i=75, while keeping K=50 and all the initialization parameters same as above. The experiment has been successful and it proves that further optimization is possible by tuning the values of η and λ accordingly. The results are visible in Table 8 below:

Dimension	Train RMSE	Test RMSE	Train MAE	Test MAE	Required iterations
50	0.6937	0.8509	0.5440	0.6645	63

Table 8: Results of Matrix Factorization approach for K=50, η=0.001, λ=0.01, i=75.

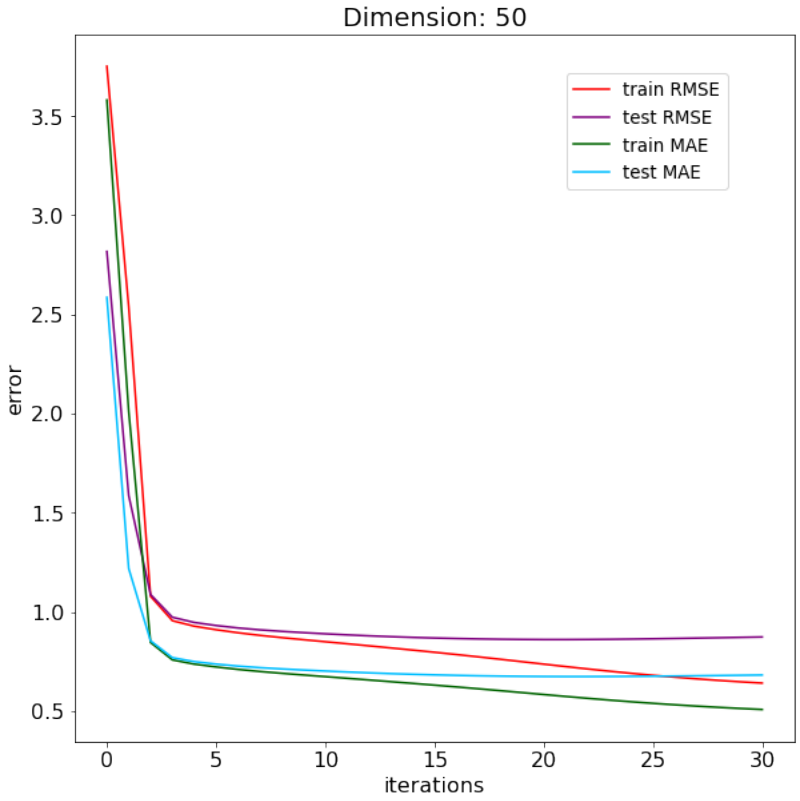

Figure 2: Graphic representation of the lowest errors on the train and test set for K=50, λ=0.05, η=0.005.

The total running time of these experiments on the Matrix Factorization approach was around 230 minutes (around 4 hours), which is much better than the UV decomposition one.

Time and space complexity

For the Global Average, the space complexity is O(1) as we need to store only two numbers (sum of ratings and the count of the ratings). The time complexity is also O(1) as we have to scan the data only once. The time and the memory for the Average Rating per movie and the Average Rating per user are the same and depend on the number of MovieIDs (m) and UserIDs (n) respectively. For the first one is O(m) while for the second one is O(n). For the Linear Regression it does not matter whether we considerate the intercept or not. The space complexity is O(n+m) whereas the time complexity is O($n \cdot m + n \cdot m^2 + m^3 + n \cdot m + m^2$), which asymptotically is equal to O($m^2(n + m)$). The UV decomposition matrix algorithm has a space complexity of O($n \cdot m + n \cdot d + d \cdot m$), however, asymptotically, we do not take into account the last two addends as they are smaller than the n x m, so we conclude that it is equal to O(n x m). On the other hand, the time complexity is O($m^2 \cdot n \cdot d + n^2 \cdot m \cdot d$), and depending on the n or m dimension we can actually eliminate one of the two addends. For the last approach, the Matrix Factorization, the space complexity is O(n x m) like the UV decomposition approach, while the time complexity is O($v \cdot d$), where v is the number of non NaN values present in the X matrix. The results are summarized in Table 9 below.

Approach	Space complexity	Time complexity (s)
Global Average Rating	O(1)	O(1)
Average Rating per movie	O(m)	O(m)
Average Rating per user	O(n)	O(n)
Linear Regression	O(n+m)	O(m^2(n+m))
UV Decomposition Matrix	O($n \cdot m$)	O($m^2 \cdot n \cdot d + n^2 \cdot m \cdot d$)
Matrix Factorization	O($n \cdot m$)	O($v \cdot d$)

Table 9: Space and time complexity of each approach.

Conclusion and Future Work

In conclusion, out of all the approaches tested, the best one is definitely the Matrix Factorization approach, not only because it is able to achieve the lowest RMSE and MAE scores, but also because of the depth of customization possible in its parameters, which make it flexible to fit well for every kind of sparse matrix X given. However, it is important to note that such an algorithm requires a lot of computational power and time to train and to find optimal values for every parameter. In circumstances where the dataset is even bigger than the one considered in these experiments, the time and power required is not a factor one could ignore. As far as the UV decomposition algorithm is concerned, we have been able to achieve quite acceptable results. However, it would be interesting to try and experiment with finding the optimal initialization parameters for higher dimensions of d, as we believe they could perform as good as if not better than the lower dimensions. Also, experimenting with increasing the number of iterations could lead to lower RMSE and MAE scores. Finally, we can say that even the naive approaches achieve quite respectable results if we consider how simple they are to build.