Advances in Data Mining

Assignment 1
Recommender Systems

Authors:	*Student number:*
Freek van Geffen	s2633256
Justin Kraaijenbrink	s2577984

Teacher: Dr. W.J. Kowalczyk

October 15, 2020

Introduction

Nowadays, customers have the possibility to view and buy millions of products offered by electronic retailers and content providers. The main goal of the retailers and providers is to help all customers find their desired product or service. However, customers can suffer from these huge assortments of products or services. Therefore personalized recommendations are needed and recommender systems are used to generate those recommendations.

There are several recommendation algorithms, some of them are more naive, such as an average rating per user, and other approaches use more advanced techniques, such as matrix decomposition. In this report, the performance of several recommendation algorithms will be compared and tested using the famous MovieLens 1M dataset, a commonly used dataset to test recommender systems. The data used in this report are retrieved from `http://grouplens.org/datasets/movielens/`.

This report is organized as follows. First we will discuss the data, the implemented algorithms, and the study design. Subsequently, the results will be shown. And finally, the results will be discussed and a conclusion is given.

Methodology

MovieLens 1M Dataset

The Movielens 1M dataset contains 1,000,209 ratings given to 3,706 movies by 6,040 users. Table 1 shows that each user rated at least 20 movies, and most of the users rated less than 250 movies. There is one movie, which is rated only one time. The most rated movie is rated 3.428 times. So we can conclude that some users have rated movies much more than other users, and some movies are rated much more than other movies.

Table 1: Descriptive statistics about the number of ratings given per user and the number of ratings received per movie.

	Mean	Std	min	25%	50%	75%	max
User	165.60	192.75	20	44	96	208	2314
Movie	268.89	384.05	1	33	123	350	3428

Figure 1 shows the distribution of the given ratings. The most given rating is four and the least given rating is one. Overall, the ratings three, four and five are given more often than one and two.

Study Design

In this research, the main interest is the accuracy of the recommender systems. The accuracy is measured using the 5-fold cross-validation scheme. For this scheme, the MovieLens dataset is randomly divided into five approximately equally sized parts, using a specific random seed for reproducibility. Subsequently, the model is trained while one of the parts is not included in the train data. The left out part (test data) is used to test the model, and the accuracy is measured. This process is executed five times with different train data and test data. Then, the

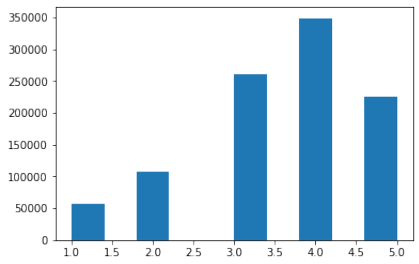

Figure 1: Histogram of the given ratings.

average over the accuracy measures are computed and this is considered as a good estimate of the accuracy. The overall runtime of the 5-fold cross validation is also measured, where runtime is based on a Lenovo G50-70, with 8GB RAM and a Intel Core i7-4500 CPU processor.

The accuracy is measured using the root mean squared error (RMSE) and the mean absolute error (MAE).

$$RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(R_k - \hat{R}_k)}$$

$$MAE = \frac{1}{n}\sum_{k=1}^{n}|R_k - \hat{R}_k|$$

where R_k is the true rating and $\hat{R}_k$ is the predicted rating.

Algorithms

In this section we provide an overview of the different recommendation algorithms that have been investigated.

Naive approaches

Global average rating: In this approach, the global average of the train data is computed and the predicted rating for every observation in the test data is the global average.

Average user rating: This approach has two steps. First, the model computes the global average and the average rating of each user in the train data. Then, for every observation in the test data, it looks whether the user has a known average user rating. If the average user rating is known, the average user rating of that specific user is given as prediction. Otherwise, the global average rating is given as prediction.

Average movie rating: This approach is comparable with the average user rating approach, but this time the average movie ratings are computed instead of the average user ratings. The model again checks whether the average movie rating is known. When known, the average movie rating is given as prediction. Otherwise, global average rating is given as prediction.

Linear regression model: In this approach, a linear model is computed to predict the rating.

$$\hat{R}_{user,movie} = \alpha * avg_{user} + \beta * avg_{movie} + \gamma \tag{1}$$

Based on the train data, the parameter θ, which is the combined set of α, β and γ, is estimated using least squares approximation, which minimizes the sum of the squares of the residuals of the following equation.

$$\mathbf{y}_{n\times 1} = \mathbf{x}_{n\times 3} * \theta_{3\times 1} \tag{2}$$

where n represents the number of observations in the train data, $\mathbf{y}$ represents the ratings in the train data, and where the columns of $\mathbf{x}$ represent respectively, the average user rating in the train data, the average movie rating in the train data, and a column with ones.

θ can be estimated through solving

$$\hat{\theta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \tag{3}$$

where $\hat{\theta}_1$ is α, $\hat{\theta}_2$ is β and $\hat{\theta}_3$ is γ.

Thus, first this approach computes the linear model parameters, the global average rating, the average user ratings, and the average movie ratings based on the train data. Then, predictions ($\hat{R}$) are made for the test data, using equation 1. This approach contains some fall-back rules, needed when the average user or the average movie rating is not available. First of all, if the average user rating is unknown, the global average is used to replace avg_{user} in the linear model. Secondly, if the average movie rating is unknown, the global average is used to replace avg_{movie} in the linear model. And lastly, if the average user rating and the average movie rating are both unknown, the prediction is the global average, which is equal to replacing avg_{user} and avg_{movie} with the global average rating in equation 1.

Note that the regression approach can generate ratings that are greater than five or smaller than one are invalid ratings. Therefore, as final step in this approach, ratings greater than five are adjusted to five, and ratings smaller than one are adjusted to one.

UV-Decomposition algorithm

UV-decomposition is a specific case of *singular value decomposition*, and makes use of the principle of dimensionality reduction. The overarching rationale is that we can approximate a matrix $R \in \mathbb{R}^{n\times m}$ by the multiplication of two matrices $U \in \mathbb{R}^{n\times d}$ and $V \in \mathbb{R}^{d\times m}$, with n the number of users, m the number of movies and d the number of dimensions. The dimensions d are somewhat abstract, but we can think of it as a number of features that captures the relationship between users and movies. For example, genre possibly determines to a certain extent whether someone likes a movie or not.

The UV-decomposition algorithm is concerned with finding appropriate values for U and V, such that the non-blank elements of the sparse user-movie-rating matrix R are as close as possible to the corresponding elements in matrix $\hat{R} = UV$. This 'closeness' is measured by the RMSE and the update of values in both U and V takes place in an element-wise manner. We implemented the algorithm as follows:

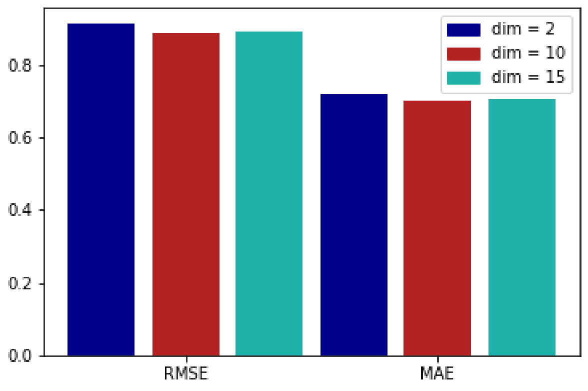

Figure 2: 5-fold cross-validation average RMSE and MAE for $d = \{2, 10, 15\}$.

1. Initialize U and V with all elements being $\sqrt{a/d}$, where a is the average rating of R and d is the number of dimensions we pick
2. Randomly pick an element in U and V and update using formulas (4) and (5):

$$u_{rs} = \frac{\sum_j v_{sj}(r_{rj} - \sum_{k \neq s} u_{rk} v_{kj})}{\sum_j v_{sj}^2} \tag{4}$$

$$v_{rs} = \frac{\sum_j u_{ir}(r_{is} - \sum_{k \neq r} u_{ik} v_{ks})}{\sum_j u_{ir}^2} \tag{5}$$

3. Calculate the RMSE with updated U and V
4. While the improvement in RMSE exceeds a certain threshold $\epsilon = 0.01$, repeat steps 2. and 3.

There are two important notes to make here. First of all, we should mention that the final values of U and V are dependent on the order in which the elements are updated. This randomness can induce accurate results on the training data, but rather poor results when U and V are used to evaluate results on test data. To prevent ourselves from this kind of overtraining without excessively increasing computation times, we decided to run the above procedure twice, and average U and V over these two iterations. Another approach to avoid overfitting, is by not being too strict on the stopping criterium. We therefore set $\epsilon = 0.01$.

The second note is that the choice of d is somewhat arbitrary. Although looping over a grid with many choices for d was computationally too expensive, we did use 5-fold cross-validation to investigate which $d \in \{2, 10, 15\}$ yielded the lowest RMSE and MAE on the test data. From Figure 2 we can conclude that $d = 10$ is an appropriate choice.

Matrix Factorization

The idea behind matrix factorization is that ratings are represented as a sparse matrix $\mathbf{R} \in \mathbb{R}^{i \times j}$, where i represents the number of users and j represents the number of movies. Some ratings are unknown and are represented in the sparse matrix as zeros. Matrix factorization decomposes the sparse ratings matrix into two matrices of reduced dimensionality (f).

$$\mathbf{R} = \mathbf{U}\mathbf{M}^T \tag{6}$$

The first step of the algorithm is to randomly generate matrices $\mathbf{U} \in \mathbb{R}^{i \times f}$ and $\mathbf{M} \in \mathbb{R}^{j \times f}$. These matrices are used to predict a rating for every known rating in the train data. Note that predictions higher than five will be rounded to five, and predictions lower than one will be rounded to one. Next, the sum of squared errors is computed.

$$SE = \sum (r_{i,j} - \sum_{f=1}^{F} u_{if} m_{jf}^T)^2 \tag{7}$$

Subsequently, the sum of the squared error is minimized using a simple gradient descent method to find a local minimum. By computing the gradient of the squared error, the following updates rules in the opposite direction of the gradient can be derived.

$$u'_{if} = u_{if} + \eta(2e_{ij} * m_{jf} - \lambda * u_{if}) \tag{8}$$

$$m'_{jf} = u_{if} + \eta(2e_{ij} * m_{jf} - \lambda * u_{if}) \tag{9}$$

Where η represents the learning rate and λ the regularization term.

Thus in summary, first the latent factor matrices $\mathbf{U}$ and $\mathbf{M}$ are randomly generated, using a specific random seed to ensure reproducibility. Then, loop over each known rating $r_{i,j}$ in the train data and compute the error $e_{i,j}$ and the gradient of squared error $e_{i,j}^2$, and update the ith row of $\mathbf{U}$ and the jth row of $\mathbf{M}$. Subsequently, compute the (train) RMSE using equation 6 to compute $\hat{\mathbf{R}}$. This loop is repeated until the maximum number of iterations is reached, or when the (train) RMSE does not decrease during two iterations.

Note that the matrix factorization uses an different approach to create the train data and test data, because the sparse matrix R should always keep the same number of rows and columns. Therefore, ratings that are used as test data are modified to zero when training the model, and ratings that are used as train data are modified to zero when testing the model. Subsequently, the RMSE and MAE are calculated using only the non zero ratings in $\mathbf{R}$ and $\hat{\mathbf{R}}$.

Instead of using grid search to find the optimal parameters, we have selected the parameters based on the reported parameters on the MyMedialite website, in order to compare the performance of our algorithm with the published performances. The following parameters are used. The number of factors is ten, the maximum number of iterations is 75, the regularization term (λ) is 0.05 and the learning rate (η) is 0.005.

Results

5-fold cross-validation was used to assess the performance of the different approaches. RMSE and MAE were used as measures of accuracy, and for each of the six different approaches we computed these quantities on both the train data as well as the test data. Results are in concordance with the results reported at `http://mymedialite.net/examples/datasets.html` and are presented in Table 2.

Table 2: Performance of the algorithms using 5-fold cross-validation.

	Train		Test		
Algorithm	RMSE	MAE	RMSE	MAE	Runtime (in s)
Global average	1.117	0.934	1.117	0.934	3.053
Movie average	0.974	0.778	0.979	0.782	12.907
User average	1.028	0.823	1.036	0.829	13.271
Regression	0.915	0.725	0.924	0.732	49.748
UV decomposition	0.836	0.656	0.887	0.701	1485.405
Matrix Factorization	0.093	0.186	0.880	0.691	8551.792

Naive approaches From Table 2 we observe that the global average comes off badly in terms of RMSE and MAE, although it is computationally the least expensive. This makes perfect sense, since the global average is the simplest predictive measure. Using the movie averages or user averages already improves on the predictive performance and regression achieves even better results. This is exactly what one could have expected. Compare it to a more classical statistical problem, where a response variable Y is predicted from one or several predictors. We could use a mean model, where every predicted value of Y obtains the same value, e.g. the global average. These predictions will be quite meager and could be improved by adding a predictor to the model, which is equivalent to using either the movie or user average. Although the results will ameliorate, they are still likely to be mediocre. Adding more predictors to the model (equivalent to the regression approach) improves the prediction capacity even more.

For all the naive approaches, runtime of the cross-validation procedure is $O(N)$, so the computing time is linear with the size of the set of ratings. If we would have used half of the ratings, runtime would have been halved. The required memory for the global average approach is O(1) (constant). For the user and movie average approaches, the required memory depends on the number of users n and number of movies m. For example, for the user approach every user mean should be calculated, which requires storage of n sums and n counts. Similar reasoning holds for the movie averages. For the user and movie average approach, we denote $O(n)$ and $O(m)$, respectively. Memory for the regression approach depends on both n and m, so $O(n + m)$.

UV-Decomposition In terms of accuracy measures, the UV-decomposition algorithm outperforms the naive approaches. This comes at the cost of being computationally more expensive, with a runtime 17 times the runtime of the regression approach. Figure 3 shows the average train and test RMSE and MAE during the cross-validation procedure. We observe that the accuracy on the test data is slightly worse compared to the train data, although this difference is not disproportional to what we could expect. From this we conclude that overtraining is out of question.

Figure 4 presents for each fold of the cross-validation procedure the development of train accuracy (left panel: RMSE, right panel: MAE).

We see that the algorithm converges with only six iterations quite fast. Furthermore, the training procedure is fairly stable, with for each fold more or less the same RMSE and MAE at each iteration. Note that the number of iterations would have increased when we had been

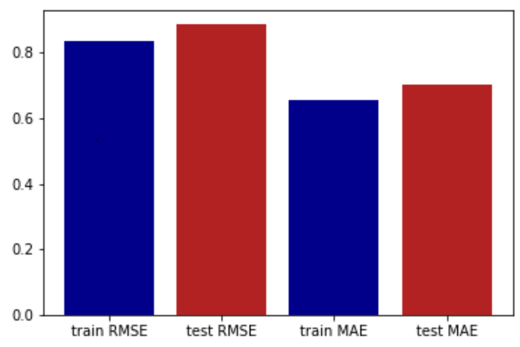

Figure 3: Average RMSE and MAE values for the UV-decomposition algorithm on both the train and test data.

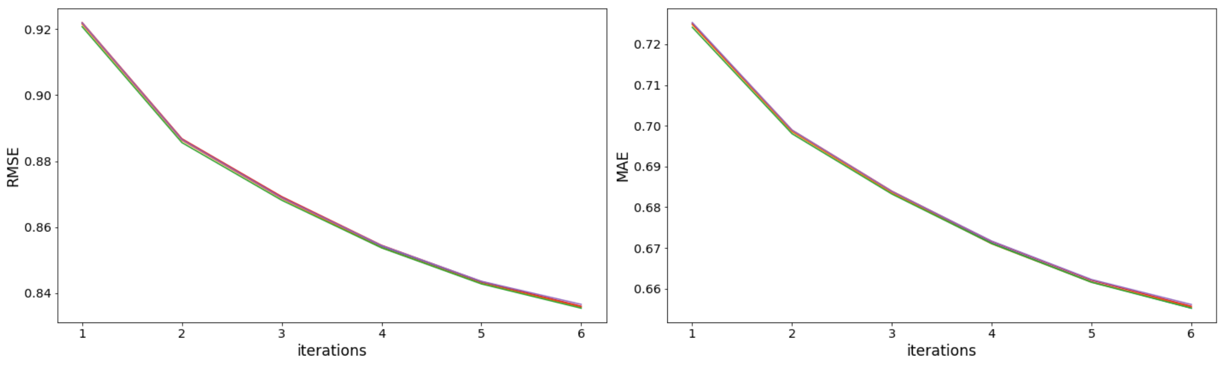

Figure 4: Development of the train accuracy (in terms of RMSE and MAE) of the UV-decomposition model.

stricter on the stopping criterion ϵ. The required memory for the UV-decomposition procedure, where we store matrices $U \in \mathbb{R}^{n\times d}$, $V \in \mathbb{R}^{d\times m}$ and $\hat{R} \in \mathbb{R}^{n\times m}$ is $O(nd + dm + nm)$. Cost in terms of runtime depends on the stopping criterion ϵ, since it determines for how many iterations i the algorithm will run. Generally, we express the UV-decomposition runtime as $O(ind)$, where we assume $n > m$ (if $n < m$, it should be $O(imd)$. When we decide to run the algorithm multiple times, say k, and average the results to prevent from overfitting, runtime is $O(imdk)$.

Matrix Factorization The matrix factorization model is the best performing algorithm based on the accuracy measures. Table 2 shows that the train accuracy measures are much better than the train accuracy measures of the other algorithms. The test accuracy measures of the matrix factorization algorithm are slightly better than the test accuracy measures of the UV decomposition algorithm.

Nevertheless, these improvements in accuracy measures come with a huge increase of computation time. The matrix factorization algorithm has a runtime more than 5 times the runtime of the UV decomposition algorithm. The higher computation time is caused by the higher number of iterations which is used in the matrix factorization algorithm. Figure 5 shows the development of the train accuracy per iteration. Figure 5 shows that a lot of iterations are necessary, since the (train) RMSE does decrease during two consecutive iterations and therefore the maximum number of iterations is executed.

The required memory for the matrix factorization is comparable with the memory for the UV-decomposition. In matrix factorization we store matrices $U \in \mathbb{R}^{i \times f}$, $M \in \mathbb{R}^{j \times f}$ and $\hat{R} \in \mathbb{R}^{i \times j}$ so the required memory is $O(if + jf + ij)$. Cost in terms of runtime is expressed as $O(Iif)$, where we assume $i > j$ (if $i < j$, it should be $O(Ijf)$, and I represents how many iterations the algorithm will run, which depends on the stop criteria and maximum number of iterations.

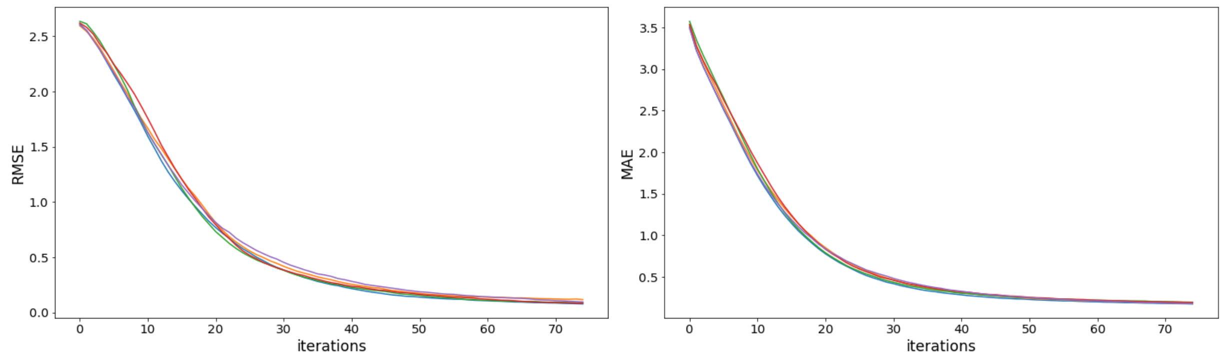

Figure 5: Development of the train accuracy (in terms of RMSE and MAE) of the matrix factorization model per iteration.

Conclusion

In this report we first presented an introduction to the problem of building recommendation systems. We provided an extensive description of the algorithms that we have used and how we implemented them, as well as the choices we made with regard to parameters we used in the simulations. We finished with the results on performance of the different algorithms in terms of accuracy combined with remarks on runtime and memory costs.

Based on the presented results, we conclude that it is possible to use naive approaches, such as using the global average, movie average, user average or using a combination of the latter two. Although they are outperformed by more advanced methods in terms of accuracy, they perform pretty well computationally wise. Especially the regression approach yield quite an acceptable RMSE and MAE, with an acceptable runtime as well. When size of the dataset increases and computation costs play a major role in deciding which algorithm to use, the regression approach should be seriously considered. Nonetheless, in most cases we will mainly be concerned with predictive capacity. In such cases Matrix Factorization would be the favorable choice. A final comment is in place here, because it would be very interesting to investigate whether different accuracy measures (such as more basic measures like the percentage of correctly predicted ratings) would results in the same conclusion. Altogether, we are happy that are results are in line with previous findings and that we succeeded in implementing several algorithms for the use of building recommendation systems!