Advances in Data Mining - Assignment 1

Recommender systems

Abstract

We implement recommender systems on the MovieLens 1M dataset (Harper and Konstan, 2015). We analyze the accuracy of these systems and use 5-fold cross-validation to verify our results. We begin by implementing several naive approaches: the global average rating, the average rating per item, the average rating per user and the linear regression on the linear combination of the previously named averages. We find that our results are very similar to the results of MyMediaLite. As expected the global average has the highest RMSE and MAE of 1.12 and 0.93 on the test set. The linear regression method is best naive approach and of the naive approaches it reaches the lowest RMSE and MAE of 0.92 and 0.73 on the test set. Consecutively, we implement matrix factorization using Gradient Descent with regularization. We test various initializations of this algorithm and for the best initialization we find that matrix factorization retrieves a RMSE of 0.856 and an MAE of 0.674. We do however find that methods with the highest accuracy often also have the highest complexity and that these methods might not be easily applied to much larger datasets.

1 Introduction

Being able to predict what content users might like, is of great interest to companies such as Amazon, Netflix, Reddit and many others. These companies have implemented sophisticated systems to make these predictions automatically. These types of systems are collectively known as recommender systems. The importance of the quality of these systems was demonstrated by the one million dollar prize that Netflix would award to the people that could improve the accuracy of the Netflix recommendation system by 10%[1](Netflixprize). The competition began in October 2006 and ended almost three years later, when the first team reached the 10% improvement mark (Koren, 2009).

There are two main techniques to make recommendations for a given user: content-based and collaborative filtering. Content-based filtering compares the description of an item to the profile of a user. The description can consist of a set of features that characterize the item. These descriptions of items that were previously rated by the user construct the profile of the user. The candidate items with descriptions that are most similar a user's profile, will be recommended to that user. Collaborative filtering tries to find users with similar interests. Recommendations are made based on what users similar to a target user have liked before. If person A has liked three different movies and person B has liked two of those as well, but has not seen the third movie, chances are that person B will like that third movie more than a randomly chosen movie.

For this project we will build a recommender system for the MovieLens 1M dataset (Harper and Konstan, 2015; Grouplens) [2] in the form of `<userID::movieID::rating>`. We will implement several recommendation algorithms and estimate their accuracy. We will start by implementing naive approaches, where we take global averages, user averages, movie averages and a linear combination of the last two averages. Consecutively we implement matrix factorization with Gradient Descent and regularization and analyze its results. In section 2 we describe our data and methods used, in section 3 we describe the results of the naive approaches and matrix factorization and in sections 4 and 5 we provide a conclusion and discussion.

[1] https://www.netflixprize.com/

[2] http://grouplens.org/datasets/movielens/

2 Method

2.1 Data

The dataset used for this assignment is the MovieLens 1M dataset (Harper and Konstan, 2015). This set contains about 1.000.000 ratings (a number between 1 and 5) given to about 3.950 movies by 6.040 users. Additionally, some information is provided about movies (genre, title, production year) and users (gender, age, occupation), but these are not used for this research. This dataset is about 1/100th the size of the dataset used for the Netflix prize.

2.2 Naive approach

There are several naive approaches possible to predict the rating ($\hat{R}_{ui}$) a given user would give to a given item that has not been rated yet by that user. The simplest of these approaches is that every unseen item would get a rating that is the same as the mean of all ratings over the whole dataset.

$$\hat{R}_{global} = \overline{R_{all}} \tag{1}$$

As this is far too general to be able to make predictions for a single user-item combination, we can use the average rating of a single user ($\overline{R_u}$) to predict the rating of an unseen item for that particular user (R_u). Similarly, the average rating for a single item ($\overline{R_i}$) can be used to do the same. This gives us:

$$\hat{R}_u(user, item) = \overline{R_u} \tag{2}$$

$$\hat{R}_i(user, item) = \overline{R_i} \tag{3}$$

Equations 2 and 3 can be combined to give a linear combination of the ratings by that user and for that item. We then get the following equation:

$$\hat{R}_{ui}(user, item) = \alpha\hat{R}_u + \beta\hat{R}_i + \gamma \tag{4}$$

where the parameters α, β and γ can be estimated using linear regression.

2.3 Matrix factorization

Another way to recommend items to user is called matrix factorization and was originally proposed by Simon Funk in a blog post in 2006. He proposed to factorize the user-item matrix as the product of two matrices of lower dimensions. The first matrix has a row for each user, while the second matrix has a column for each item. Both of these matrices use the same number of latent factors. These factors are not directly observed variables, but inferred from other variables and can be genres in the case of movie recommendations. In matrix form, this can be described as follows:

$$\hat{\boldsymbol{R}} = \boldsymbol{U}\boldsymbol{M}, \quad \underset{u\times i}{\begin{bmatrix} \hat{R}_{11} & \dots & \hat{R}_{1i} \\ \vdots & \ddots & \vdots \\ \hat{R}_{u1} & \dots & \hat{R}_{ui} \end{bmatrix}} = \underset{u\times f}{\begin{bmatrix} U_{11} & \dots & U_{1f} \\ \vdots & \ddots & \dots \\ U_{u1} & \dots & U_{uf} \end{bmatrix}} \underset{f\times i}{\begin{bmatrix} M_{11} & \dots & M_{1i} \\ \vdots & \ddots & \dots \\ M_{f1} & \dots & M_{fi} \end{bmatrix}} \tag{5}$$

where $\hat{\boldsymbol{R}}$ is the $u \times i$ predicted ratings matrix, $\boldsymbol{U}$ the $u \times f$ user-feature matrix and $\boldsymbol{M}$ the $f \times i$ movie-feature matrix. In this case f is the number of features and u and i are the number of users and item respectively. The row $U_u = \begin{bmatrix} U_{u1} & \dots & U_{uf} \end{bmatrix}$ and column $M_i = \begin{bmatrix} M_{1i} & \dots & M_{fi} \end{bmatrix}^\top$ are vectors of features for a given user u or a given item i respectively. To predict a certain rating $\hat{R}_{ui}$, we simple multiply these vectors:

$$\hat{R}_{ui} = U_u^\top M_i = \begin{bmatrix} U_{u1} & \dots & U_{uf} \end{bmatrix}^\top \cdot \begin{bmatrix} M_{1i} \\ \vdots \\ M_{fi} \end{bmatrix} \tag{6}$$

To find the best values of $\boldsymbol{U}$ and $\boldsymbol{M}$ and thus the best predicted ratings, we use the algorithm described in Takacs et al. (2007). We initialize the 'weights' matrices $\boldsymbol{U}$ and $\boldsymbol{U}$ by random sampling from the normal distribution $\mathcal{N}\left(\sqrt{\frac{\hat{R}_{global}}{f}}, \frac{1}{f^2}\right)$. Furthermore we construct the true ratings matrix $\boldsymbol{R}$ which contains all existing ratings from the MovieLens 1M data set. The elements of this matrix for which no rating exists, are set to value

zero. We update the weights matrices $\boldsymbol{U}$ and $\boldsymbol{M}$ using the Gradient Descent algorithm. Gradient Descent can be used for finding the (local) minimum of a function and is based on taking steps proportional to the gradient of that function in the direction of the minimum.

For each existing rating about movie i and rated by user u, we calculate the difference e_{ui} between the predicted rating $\hat{\boldsymbol{R}}$ (see equation 5) and the true rating $\boldsymbol{R}$

$$e_{ui} = R_{ui} - \hat{R}_{ui} \tag{7}$$

and update the uth row of $\boldsymbol{U}$ (U_u) and ith column of $\boldsymbol{M}$ (M_i) using the Gradient Descent algorithm as follows

$$U_u = U_u + \eta \cdot (2e_{ui} \cdot M_i^\top - \lambda \cdot U_u), \qquad \text{with } U_u = \begin{bmatrix} U_{u1} & \dots & U_{uf} \end{bmatrix} \tag{8}$$

$$M_i = M_i + \eta \cdot (2e_{ui} \cdot U_u^\top - \lambda \cdot M_i) \qquad \text{and } M_i = \begin{bmatrix} M_{1i} \\ \vdots \\ M_{fi} \end{bmatrix} \tag{9}$$

where η is the learning rate and λ is the regularization parameter. The regularization parameter is a measure against overfitting. To be able to compare the results of this research with previous results we use a learning rate of $\eta = 0.005$ and a regularization parameter of $\lambda = 0.05$. Furthermore we use 75 iterations over the complete training set, and we use $f = 10$ factors. After training, we post-process our predictions by changing the values that are lower than one to 1 and the values that are higher than five to 5. An overview of this algorithm can be found in figure 1.

Algorithm 1 Gravity Recommender system (Takacs et al., 2007)

initialize f, num_iter, η, λ, n_folds
for 0 to n_folds **do**
  Initialize $\boldsymbol{U}$ and $\boldsymbol{M}$ by random sampling from $\mathcal{N}\left(\sqrt{\frac{\hat{R}_{global}}{f}}, \frac{1}{f^2}\right)$
  for 0 to num_iter **do**
    for all existing ratings in training set **do**
      $e_{ui} \leftarrow R_{ui} - \hat{R}_{ui}$
      $U_u \leftarrow U_u + \eta \cdot (2e_{ui} \cdot M_i^\top - \lambda \cdot U_u)$
      $M_i \leftarrow M_i + \eta \cdot (2e_{ui} \cdot U_u^\top - \lambda \cdot M_i)$
  $\hat{\boldsymbol{R}} \leftarrow \boldsymbol{U} \cdot \boldsymbol{M}$
  $\hat{\boldsymbol{R}} \leftarrow \text{clip}(\hat{\boldsymbol{R}}, 1, 5)$ *//post-processing*
  $RMSE \leftarrow \sqrt{\frac{\sum_{ui}^{n}(\boldsymbol{R}_{ui} - \hat{\boldsymbol{R}}_{ui})^2}{n}}$
  $MAE \leftarrow \frac{\sum_{ui}^{n} |e_{ui}|}{n}$
Average RMSE and MAE over the number of folds

2.4 Accuracy

To measure the accuracy of the different methods, we use two different estimators, the Root Mean Squared Error, RMSE, and the Mean Absolute Error, MAE. Defining the predicted value as $\hat{y}_i$ and observed value as y_i, we can define these measures as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{10}$$

$$MAE = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n} \tag{11}$$

To verify that our results are reliable, we make use of 5-fold cross-validation. This means that our dataset was shuffled and partitioned into 5 equally sized subsets. The different algorithms are trained on 4 out of 5 parts of the total dataset and subsequently tested on the final 1/5 part of the set that was not used in the training process. This is done for each of the 5 subsets of the data, eventually taking the average error of these 5 different results. The random seed that we use for dividing the data in 5 subsets and for initializing $\boldsymbol{U}$ and $\boldsymbol{M}$ is 9294.

3 Results

3.1 Naive approaches

The results of our four different naive approaches are given in Table 1. The error measures for the RMSE and MAE are reported for the training and test sets using the four different naive approaches. The values are the averages of the 5 different folds used. It is clearly visible that the error measures decrease and the run time increases with an increase of the information used to predict the ratings. As there are more users than movies, the run time for the user average is larger than for the item average. One might also suspect that the error measures for the user averages would be lower. This is not the case however, which is due to the fact that there are on average more ratings per movie than per user (by a factor of ~ 1.528), giving a more accurate prediction.

We also compared our results to the results posted on the MyMediaLite (MML) website [3]. The results for the linear approach were not reported by MML. There are only small differences between the error measures we reported and those from MML. These differences could be due to a different random seed used, but when trying different seeds, we achieved the same results. Therefore, there probably is a small difference in the dataset used or in the implementation of the algorithm. The main trend of the results is the same however, so these differences were not deemed a problem.

Algorithm	**RMSE**			**MAE**			**Run time**
	training	test	*MML test*	training	test	*MML test*	(s)
Global avg.	1.117	1.117	*1.117*	0.9339	0.9339	*0.934*	0.88
User avg.	1.028	1.035	*1.036*	0.8227	0.8290	*0.827*	129.4
Item avg.	0.9742	0.9794	*0.983*	0.7783	0.7823	*0.783*	98.8
Lin. regression	0.9146	0.9243	*NA*	0.7251	0.7327	*NA*	184.7

Table 1: Comparison of the root mean square error (RMSE) and mean absolute error (MAE) for the training and test sets using the four different naive approaches. The values are the averages after 5-fold cross-validation. Additionally the results posted on the MyMediaLite (MML) website for the four different naive approaches are displayed. The results for the linear regression approach were not reported by MML. The error measures decrease with the increase of information used to predict the ratings, which is to be expected. The run time increases with the increase of information used (as there are more users than movies). The error measures on the test sets are slightly higher than on the training sets. There are only small differences between our error measures and those reported by MML.

3.2 Matrix factorization

The results of matrix factorization for the default initializations 'MF A' and 'MF F' are given in Table 3. The details of these initialization can be found in Table 2 and they can be compared with the results of MyMediaLite. For 'MF A', the update rules from Equations 8 and 9 as described in Takacs et al. (2007) are used and additionally a learning rate of $\eta = 0.005$ and regularization parameter of $\lambda = 0.05$. For 'MF F' the same update rules have been used and we doubled the regularization rate and halved the learning rate. This equals to the following update rules, with the same learning rate ($\eta = 0.005$) and ($\lambda = 0.05$) as in 'MF A'.

$$U_u = U_u + \eta \cdot (e_{ui} \cdot M_i^\top - \lambda \cdot U_u), \qquad \text{with } U_u = \begin{bmatrix} U_{u1} & \dots & U_{uf} \end{bmatrix} \tag{12}$$

$$M_i = M_i + \eta \cdot (e_{ui} \cdot U_u^\top - \lambda \cdot M_i) \qquad \text{and } M_i = \begin{bmatrix} M_{1i} \\ \vdots \\ M_{fi} \end{bmatrix} \tag{13}$$

Notice that the difference between Equations 8 and 9, and 12 and 13 is that the factor 2 in front of e_{ui} is removed. The results of 'MF F' are very similar to the results of MyMediaLite, much more so than the results of 'MF A'. This is why we think that they might have used the above mentioned update rules (Equations 12 and 13) instead of the equations described in Takacs et al. (2007).

Besides 'MF A' and 'MF F', we have experimented with different initializations to investigate which initialization is best. The details of these initializations can again be found in Table 2. The results of these different intializations can be found in Table 4. The average error over the five folds per training iteration are displayed in Figure 1. We observe that 'MF B' and 'MF C', where the regularization parameter was doubled and the learning rate was halved respectively, lead to a higher accuracy on both the test and training set than the

[3] http://www.mymedialite.net/examples/datasets.html

default initialization 'MF A'. It could therefore be of interest to test if other values of the parameters η and λ would work even better.

The results of using a different number of factors 'MF D' and 'MF E' can be observed in Figures 1d and 1e. A much smaller difference between the errors on the training set and on the test set can be observed for using 5 factors instead of 10. The difference between the training set and test set increases a lot when using 20 factors instead of 10. This can be explained by the fact that it is easier to fit the training data with more factors, lowering the error on the training set and therefore increasing the difference between the errors on the training set and on the test set. When we used 20 factors, however, the algorithm clearly overtrained itself, as can be seen in Figure 1e. After a certain amount of iterations, the error on the training set keeps decreasing, whereas the error on the test set increases again. For the RMSE, the minimum error on the test set was reached after 16 iterations (at 0.8692) and for the MAE, this was reached after 20 iterations (at 0.6811). Even though the training errors for the initialization with 20 factors are by far the smallest, this is not reflected in the errors on the test set, as these are higher than all other after 75 iterations. It seems that the amount of 10 factors is therefore a well-chosen middle ground between using more factors to fit the data better, but not overfitting it.

Initialization	# factors	regularization λ	learning rate η
MF A	10	0.05	0.005
MF B	10	**0.1**	0.005
MF C	10	0.05	**0.0025**
MF D	**5**	0.05	0.005
MF E	**20**	0.05	0.005
MF F	10	**0.1**	**0.0025**

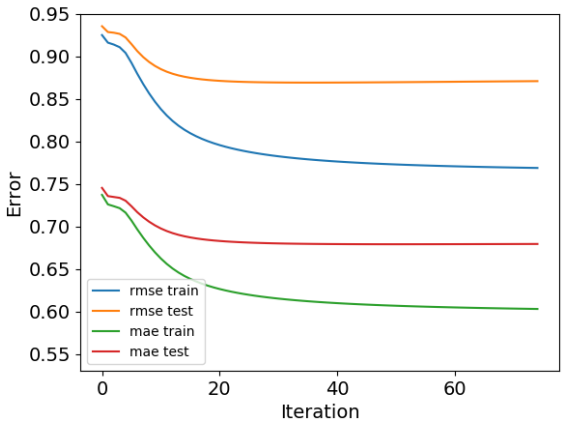
Table 2: Parameters used for the different initializations of the Matrix Factorization algorithm used in the research. The number of iterations was kept the same in all versions at 75. The first row indicates our default initialization, whereas the bold numbers in the other rows indicate the changed parameter for that particular initialization.

Algorithm	**RMSE**			**MAE**			**Run time**
	training	test	*MML test*	training	test	*MML test*	(s)
MF A	0.7687	0.8707	0.857	0.6029	0.6793	0.675	10962
MF F	0.7851	0.8561	0.857	0.6212	0.6744	0.675	10905

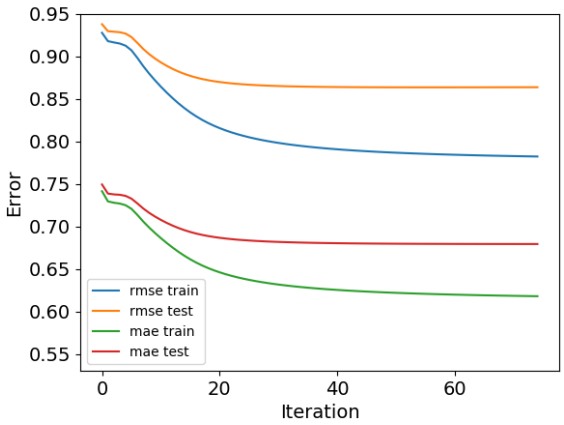
Table 3: Comparison of the root mean square error (RMSE) and mean absolute error (MAE) for the training and test sets using the initializations of 'MF A' and 'MF F'. The values are the averages after 5-fold cross-validation. Additionally the results posted on the MyMediaLite (MML) website for the Matrix Factorization algorithm are reported. The results from the initialization of 'MF F' are much more similar to the results by MML than from the initialization of 'MF A'.

Initialization	RMSE test	MAE test	RMSE training	MAE training	Run time (s)
MF A	0.8707	0.6793	0.7687	0.6029	10962
MF B	0.8636	0.6792	0.7821	0.6179	10898
MF C	0.8607	0.6728	0.7700	0.6052	10987
MF D	0.8757	0.6864	0.8224	0.6465	10901
MF E	0.8871	0.6887	0.6973	0.5445	10972
MF F	0.8561	0.6744	0.7851	0.6212	10905

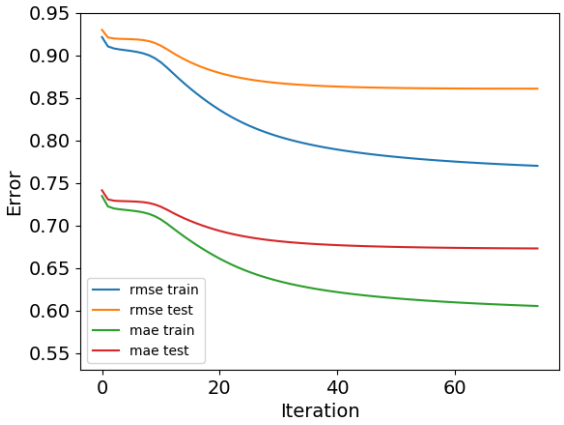
Table 4: Comparison of the root mean square error (RMSE) and mean absolute error (MAE) for the training and test sets using the different initializations of the Matrix Factorization algorithm. The values are the averages after 5-fold cross-validation. The run times are all very similar. Changing the parameters as in 'MF B, C or F' results in a positive effect on the error for the test set. The results are negative after changing the parameters as in 'MF D or E', even though the error on the training set for 'MF E' is the lowest of all.



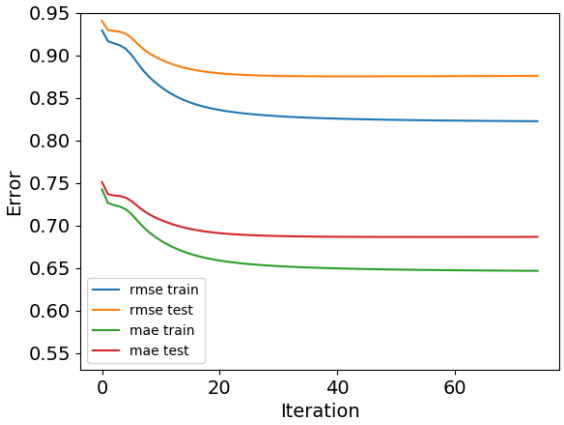
(a) Default



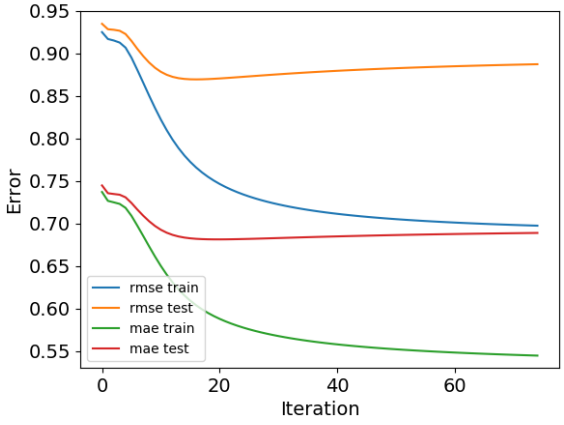
(b) Regularization $\lambda = 0.1$



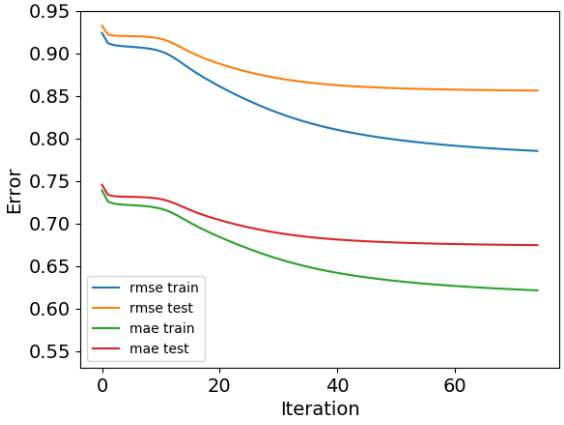
(c) Learning rate $\eta = 0.0025$



(d) Number of factors 5



(e) Number of factors 20



(f) Regularization $\lambda = 0.1$ and learning rate $\eta = 0.0025$

Figure 1: Plot of the error of matrix factorization versus the training epoch (iteration). The errors are averaged over the folds. The root mean square error (RMSE) and mean absolute error (MAE) for the training and test set are indicated with different colours. Each subplot indicates a different initialization for matrix factorization, as can be found in Tables 2 and 4. The best initialization can be found in subplot 1f. The difference in error between the training and test set is lowest for subplot 1d and highest for subplot 1e. This is to be expected with a smaller and larger amount of factors, respectively. In subplot 1e we can see that the error on the test set increases after a certain amount of iterations, as the algorithm has been overtrained on the training set.

4 Conclusion

To build a recommender system for the MovieLens 1M data set, we have implemented various algorithms. The most simple algorithms are naive approaches. In the first naive approach we have taken a global average over the total number of ratings and observed an RMSE of 1.12 and an MAE of 0.93 on the test set. This approach has the least run time and memory, but is of course a way too simple manner of giving recommendations. For the second and third naive approaches we have calculated the average over the ratings that a given user have given and the average over the ratings that a certain movie has received. These approaches are not the most smart approaches either but still received an RMSE of 1.04 and 0.98 for the test sets of the second and third naive approach respectively, and an MAE of 0.83 and 0.78 respectively. These values are already better than the first (simple) approach and are comparable to the results of the MyMediaLite website. However, improvements can still be made. The last naive approach we use is linear regression on a linear combination of the user average and the movie average. This method achieves a RMSE of 0.92 and an MAE of 0.73 on the test set.

We also used matrix factorization to predict the ratings. This was first implemented using a set of suggested variables. We also tried other values of all of these variables, except the number of iterations; this was kept the same for all experiments. Changing the number of factors by a factor of 2 upwards or downwards caused an increase in our error measures on the test set, which is negative. The suggested amount of 10 factors is therefore probably close to the optimal amount. Changing the regularization parameter upwards and the learning rate downwards decreased our error measures on the test set, which is positive. Tinkering with the exact values of these might decrease the error measures even more, but the benchmark results of the MyMediaLite website were achieved, with a RMSE of 0.856 and a MAE of 0.674 on the test set.

5 Discussion

The results of our algorithms are very similar to the results of the MyMediaLite website. However we do observe some small differences. The differences could be caused by the way different computers treat rounding errors, by the fact that different computers use different (versions of) libraries for calculations, which are amplified through various iterations or the precision of the storage of intermediate values (32-bit vs. 64-bit). The computer that was used in this research is a Dell Precision T1650 with Intel(R) Core i5-3470 CPU @ 3.20GHz.

As we have seen from the results of this research, using smarter algorithms leads to the better results. These smart algorithms however come at a price as they have increasing memory and run-time. The complexity of the algorithms used can be found in Table 5. The notation used in this table is the Big O notation (Bell; Cheatsheet; HackerRank; Python) indicating how much the time and memory complexity scale. We observe that matrix factorization has the highest time and memory complexity. Although matrix factorization achieves the best results error-wise, we must keep in mind that this method might not be best if we use much larger data sets. It will then be necessary to consider whether accuracy or time is the most important factor.

The recommender systems on the Movielens 1M dataset can still be inproved by implementing other algorithms such as biased matrix factorisation, matrix factorisation with alternating least squares or single value decomposition (SVD). The algorithms that we used in this research can be considered as supervised machine learning. It might also be interesting to use unsupervised machine learning algorithms such as clustering or deep neural networks.

Algorithm	Time complexity	Memory complexity
Global avg.	$\mathcal{O}(R)$	$\mathcal{O}(1)$
User avg.	$\mathcal{O}(R \cdot U)$	$\mathcal{O}(U)$
Item avg.	$\mathcal{O}(R \cdot M)$	$\mathcal{O}(M)$
Lin. regression	$\mathcal{O}(R \cdot (M + U))$	$\mathcal{O}(R)$
Matrix Fact.	$\mathcal{O}(F \cdot (FR + MU))$	$\mathcal{O}(MU)$

Table 5: Complexity of the recommender algorithms used. The complexity is noted in the Big-O notation indicating how the time complexity and memory complexity scale. In this case R is the number of ratings, U is the number of users, M is the number of movies and F is the number of factors. Note that the smartest and most difficult algorithms have the highest complexity but also the best results.

Bibliography

Rob Bell. A beginner's guide to Big O Notation. https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/. Accessed: 2018-09-25.

Big O Cheatsheet. Big O Complexity Chart. http://bigocheatsheet.com/. Accessed: 2018-09-25.

Grouplens. MovieLens. https://grouplens.org/datasets/movielens/. Accessed: 2018-09-25.

HackerRank. Cracking the Coding Video: Big O Notation. https://www.youtube.com/watch?v=v4cd1O4zkGw. Accessed: 2018-09-25.

F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL http://doi.acm.org/10.1145/2827872.

Yehuda Koren. The bellkor solution to the netflix grand prize, 2009.

MyMediaLite. MyMediaLite example experiments. http://www.mymedialite.net/examples/datasets.html. Accessed: 2018-09-25.

Netflixprize. Netflix Prize. https://www.netflixprize.com/. Accessed: 2018-09-25.

Interactive Python. Big O Notation. http://interactivepython.org/runestone/static/pythonds/AlgorithmAnalysis/BigONotation.html. Accessed: 2018-09-25.

Gabor Takacs, Istvan Pilaszy, Bottyan Nemeth, and Domonkos Tikk. On the gravity recommendation system. In *Proceedings of KDD cup and workshop*, volume 2007, 2007.