Advances in Data Mining 2018

Assignment 1

September 27, 2018

1 Introduction

Recommender systems are crucial in the fast-moving, data-fueled modern economy of many businesses which rely on persuading users to follow their personalized recommendations to new product items in the catalogue to generate revenue. In order for a user to utilize this system and consider the recommendations made, they have to be of high quality from the onset so to win trust of the user and encourage them to look at the recommendations for products they might have never thought of themselves. There is another side of the coin as well, where companies dont so much rely on the recommender system to generate revenue, but tying users to themselves relies entirely on the recommender system. This statement might seem contradictory, but it is in fact the kind of system studied in this report. Online monthly subscription services with unlimited use of their items once a subscription is bought, i.e. the Netflixes and Spotifys of this world, have to provide high quality recommendation for the next movie, song or other item the user might be interested in to keep the user interested in their service.

We have now established the crucial importance of recommender systems in the modern business model, but we have not yet ventured to say what kind of systems there are and how they differ. To provide the reader of this report with some basis of understanding when diving into the technicalities we will first introduce the data used here. This report will use the data from an online movie subscription service where users provide ratings in the range 1-5 for each item they have viewed, the data set contains a total of one million unique (user, item) pairs. The caveat is in the sparsity of the data, although there are approximately 6000 users who between them have rated just over 3700 items, not every user has rated every item. Though some users might have rated hundreds of items, others only rated a few. The challenge is to predict what rating users would assign to items they have not yet viewed in order to recommend those with the highest predicted ratings. Several naïve approaches can be implemented relatively easily, whereas more sophisticated approaches will be discussed at a later point in this report after a clear understanding of the former is hopefully achieved.

2 Naive Approaches

In this section we will implement four different naïve approaches to predict the rating that a certain user would give a certain movie. These approaches are all based on very elementary mathematical computations and will provide a baseline for more sophisticated recommender systems.

2.1 Concepts

The simplest approach one can think of is the computation of a global average. The prediction for every (user, item) pair without a rating is, in this clearly over-simplistic approach, the average of all (user, item) pairs which do have a rating assigned to them. We could justly call this a zeroth order approach to the problem and do not expect a very accurate prediction. The global mean is defined as

$$\text{global mean} = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{1}$$

where n is the total number of ratings and x_i the individual ratings in the dataset. Intuitively, we can already see that several specifications should be made, from these we will form the next two approaches.

First, we can imagine that some users are particularly grumpy, they might have non-satisfying jobs, suffer severe depressions or live in parts of the country with poor internet connectivity which affects their rating of our streamed items. Others might have just had a promotion, found the love of their lives or simply live on a

tropical island with nothing to complain. It is easy to imagine that the first group of users systematically rates items lower than the second group of users. This leads us to a possible improvement from the global average, perhaps we should predict ratings by taking the average of all (user, item) pairs which have assigned a rating, but only for one specific user. This is called the user mean and is similar to stone age meteorology, just as the weather tomorrow is more likely to be like today's weather than completely different, just so a user is more likely to rate an item like their average of previous items than completely different. Clearly this will work better for users which have rated large amounts of movies, than for those with only a few.

$$\text{user mean} = \frac{1}{n_{items}} \sum_{i=1}^{n_{items}} x_i, \tag{2}$$

where n_{items} is the number of items a user has rated and x_i the ratings that the user has given.

A second approach, completely disparate from the user mean, is to move ourselves into the point of view of the items. Certain items might be high quality, price winning miracles of otherworldly beauty whereas others are low quality DIY, handheld camera disappointments. As disconnected observers with nothing but user and item ID numbers, we can reasonably expect that some items will be rated higher and others lower than the global average by all users, simply because there is an intrinsic quality difference of which we are not a priori aware. This leads to another possible improvement of the global average: perhaps we should predict ratings by taking the average of all (user, item) pairs which have assigned a rating, but only for a specific item. This we will call the item mean and puts into practice the concepts explained above. It might be likely that a user rates a new item just as all other users have, on average, rated this item.

$$\text{item mean} = \frac{1}{n_{users}} \sum_{i=1}^{n_{users}} x_i, \tag{3}$$

where n_{users} is the number of users that have rated an item and x_i the ratings that an item has received.

Though, intuitively, both approaches bring improvements, it is hard to imagine that only one of them will be important. Very likely both will have a large impact on the accuracy of the predicted rating. As an example, a user with a lower than average user mean, will likely still rate certain items higher than others, which indicates that the lower user mean should be combined with the different item means to come to an accurate prediction for each (user, item) pair for this user. This insight leads us to consider how we could be implement a combination of the user and item mean to obtain a maximally accurate prediction. It stands to reason that the user mean and the item mean will likely be of unequal importance, we should perhaps assign different weights to them when making the prediction for a specific (user, item) pair. Furthermore, there could be some overall bias, which we could possibly detect it in the global average. If our users where truly balanced, giving –on average between all users– ratings uniformly distributed between 1-5, a global average of 3 should be found, an unlikely occurrence. Since our data is gathered in the US, a country where social-political bias is towards praising the individual in the hope this will increase self confidence and enhance their social-economic status in the long run, we expect to find an overall bias to our ratings such that the global average is above 3. This is further supported by a likely selection bias by the provider of the service this recommender system is built for, they have probably selected a range of items which are of above average quality, e.g. a disproportionately large number has won an award compared to the real world, so we again expect to see a bias in our global average.

Taking this all together, we need to built a model which takes the user- and item mean with different weights assigned to both and an overall bias, to predict the rating for each empty (user, item) pair, but this is nothing else than linear regression. This is our fourth naïve approach, with which we mean that it can be thought up without the need of any complex mathematical manipulations.

$$\text{Linear Regression} = \alpha * \text{user mean} + \beta * \text{item mean} + \gamma \tag{4}$$

We might now endeavour to apply these approaches and observe their effectiveness, however, that is not as straightforward as it might seem. Ideally, we would ask all users to watch a couple of movies, provide us with the ratings for those movies and compare with the prediction we have made. Unfortunately we are in no position to do so, let alone that it would jeopardise the independence of the ratings as users might very well rate differently when aware that their ratings will be used in an experiment. Instead we will partition our data in two different subsets, the test set and the training set. The training set is, clearly, meant to train our algorithms on and predict ratings for users. Once we have done this we apply our trained algorithm to the test set and see how accurate we are. This completely obliterates the need for any intermediate user input, we now treat the test set as new ratings which would have otherwise been provided by the users and test the accuracy of our predictions on these newly acquired ratings. A couple of intricacies arise, the most obvious is a mismatch between training data and test data. There will, likely, be users and items which occur only in one of the sets. A specific example is this, imagine creating the user mean for all users in the training set based solely on the ratings in the training set, when applying this to the test set to test how accurate our

predictions for ratings in the test set are, we might encounter a user in the test set which was not present in the training set, the approach we have so far would simply not be able to make any predictions for this user at all. A very simplistic work around applied in this report is to assign predictions to those users based on the global mean. There are multiple ways to implement a train/test division of data, the one used in this report is the fivefold cross validation, the data is randomly split into five equal chunks, four of those make up the training set and the fifth serves as the test set. We then repeat this procedure five times such that all five sets have been the test set once and calculate the accuracy of our predictions by taking the average over these five iterations. This now provides a reliable estimate for the accuracy we would achieve on the entire data set. Our accuracy will be calculated in two ways, the Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i - \hat{x_i}}, \tag{5}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |x_i - \hat{x_i}|, \tag{6}$$

where n is the total number of evaluated ratings, x_i are the actual ratings and $\hat{x_i}$ the predicted ratings. It can be easily seen from these two equations that the RMSE is bounded by:

$$MAE \leq RMSE \leq MAE\sqrt{n} \tag{7}$$

There are some important and clear differences between the two errors, but we will first examine the similarities. Both RMSE and MAE measure the magnitude of the error and are independent of the direction, that is saying over-predicting a user rating by a given amount is just as bad as under-predicting it by the same amount, this is clearly a property we desire. Both are also oriented such that a lower score indicates better predictions and both range from zero to infinity. Both also have the mean in their name, but this is actually where the difference is at. In the RMSE the mean is taken after squaring and summing the error, but before taking the root, this results in a quadratically larger weight for larger errors, the MAE takes the mean after taking the absolute value of the error, which is equivalent to taking the mean after squaring the error, but before summing and then also taking the root before taking the mean. In other words, if the total amount of error summed up over all instances is fixed, but the frequency distribution of the error is changed from being equally distributed over all instances, to having a large number of instances with a small error and a small number with a large error, the MAE stays the same, but the RMSE increases. Specifically, in our case, this means that the RMSE penalizes a prediction being off twice as much as another prediction, more than twice the amount, whereas the MAE will assign the same value. As a result, minimizing the RMSE will drive the predictions to having the frequency distribution of the magnitudes of errors be uniform, while minimizing the overall error, and minimizing the MAE will drive the predictions to simply have the overall amount of error minimized, which it does more effectively than the RMSE.

2.2 Results

We will now present and interpret the results obtained using the four naïve approaches described above. The RMSE and MAE for the four approaches can be found in Table 1. Certain trends can be clearly discerned from the error measures. With increasing complexity of the approach used, the RMSE exclusively decreases, the MAE decreases as well except for the linear regression approach. The latter is a striking feature we will describe in more detail later in this report. It can also be observed that the error on the training set is always smaller than the same error measure on the test set. This is entirely within expectation since the training set is the part of the data on which the error measure is minimized, after minimizing the recommendations are applied to the test set, however since we do not minimize over the test set but merely test, it is to be expected that the error here

Table 1: Prediction RMSE and MAE obtained using different predictor mechanisms on both training and test sets using fivefold cross validation.

Predictor	RMSE		MAE	
	Test	Training	Test	Training
Global Mean	1.117	1.117	0.934	0.934
User Mean	1.035	1.028	0.829	0.822
Item Mean	0.979	0.974	0.782	0.778
Linear Regression	0.924	0.915	0.854	0.837

is slightly larger. We would like to note that the difference is quite small, only the linear regression MAE differs more than 1 percent point between the training and test set, indicating that the test and train sets are indeed comparable without any large bias present in the random division of the data into these two sets. Had there been a major bias in the selection, the error measures would have differed significantly more as the training set would no longer be representative of the test set. There is one exception, which is the global mean. When looking closer at the output we receive, a slight difference between the training and test sets can be discerned in the sixth or seven digit. That these numbers are so close is not surprising, since our test and training sets are created in a random procedure and are huge ($\sim$ 200,000 and 800,000 entries) it would be staggering to find the error measure of the global means differ by more than near negligible amounts.
Returning to the difference between the RMSE and MAE error measures, we note that the RMSE for the linear

Table 2: Linear regression parameters obtained by a simple least square linear regression model imported from the sklear.linear_model package.

	Mean	Standard deviation(1σ)
α	0.782	0.0005
β	0.875	0.0004
γ	-2.352	0.003

regression model is lower than that of any of the other models, but the MAE is larger. Luckily this is easily understood by having a quick look at the documentation of the specific linear regression fitting model we have used, namely the sklear.linear_model.LinearRegression()[1] package. This model optimizes a linear regression by 'ordinary least squares' which, just like RMSE, drives the frequency distribution of the magnitude of errors to be as close to uniform as possible, as it is basically the RMSE without the root and the mean. Since the linear regression model is the only model used in this report which explicitly uses a specific minimization approach to fit its values rather than relying purely on the data, the result of this is reflected in it driving down the RMSE, but being less favourble for the MAE which puts higher priority in minimizing the overall amount of error. If one was to use a linear regression model based on minimizing the absolute error rather than the square error, the opposite would likeley be true, with a lower MAE and a higher RMSE compared to the other models.
Other interesting takeaways from these results are:

- Even for the best model the error on predictions is close to one. This is quite bad given that the rating scale only goes from 1-5. Further improvements can clearly be made here.
- The item mean is a better predictor for the rating than the user mean. This indicates that the rating for a specific (user, item) pair is more dependent on the intrinsic quality of that item than on the rating bias of a user.
- The MAE is always lower than the RMSE. Given what has been said about these two error measures above, this indicates that disproportionately large errors are made for certain (user, item) pairs compared to the average error which would be obtained from dividing the total amount of error by the number of (user, item) pairs for which a prediction is made.

2.3 Computational Strain

In this section we will provide an overview of the computation time and the required memory to run the code implemented to obtain the results in this report. After a brief overview of the main structures in our code, we will give a comprehensive scheme detailing how the time and memory required scales with the number of users and items in the data.
Our code is made up from two major pieces, the first is made up of functions which execute the user/item specific calculation of the means where the second piece governs the fivefold cross validation a calls the functions in the first piece. We will indicate time and memory usage by big-O notation, where U, M and R represent the number of users, movies and ratings respectively. The results we give here are for an ideal algorithm which would be optimized to use as little memory and time as possible, it is certainly not a direct measure for the time we need in our code.

Global Mean This results in a single number calculated from two numbers, being the sum of all entries and the amount of entries, which means that the memory required is only O(1). The time required to calculate the global mean scales with the number of ratings, since we have to loop over all ratings, so the computation time scales as O(R).

[1] see also: `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`

User Mean The user mean is equivalent to the global mean, but we check for every rating the user, therefore the memory required for one user mean is O(1), but the total memory required to store the user means for all users is O(U). To calculate all user means we ideally only loop over all ratings once and determine all possible user means, thus the time scales as O(R).

Item Mean Similar to the user mean, a single item mean requires O(1) memory, all item means require O(M) memory. The required time is again O(R) as we check all ratings in the rating matrix to check whether it is related to an item.

Linear Regression To compute this we need to solve the linear Equation 4, which requires O(R) of memory. The algorithm then loops over all ratings to minimize a least squares procedure, O(R) in time, which can also be seen from the matrix equation for linear regression where we multiply a (1x2) matrix with a (2xR) matrix, requiring O(R) time.

There are ways to optimize these parameters by only reading one (user, item) pair at a time from the data file instead of loading all the data into an array, but because of the relatively limited size of the data set, that has not been done in the code accompanying this report.

3 Matrix factorization

In this section we will implement the matrix factorization algorithm, following Takács et al. (2007). For the matrix factorization algorithm we construct a matrix X (dimension $I \times J$, with I the total number of users and J the total number of movies) where each row corresponds to a given user and each column corresponds to a given movie. We fill this matrix with all 1,000,209 available ratings. The matrix will thus be a very sparse matrix, as the number of possible (user,movie) combinations is $I \times J = 6040 \times 3706 = 22,384,240$. Thus, we only have data available for 4.47% of the entries.

3.1 Concept

The goal of matrix factorization is to use the available entries of X to try to predict the missing entries. This is done by approximating the matrix X as the product of two matrices:

$$X \approx UM, \tag{8}$$

where U is an $I \times K$ matrix and M a $K \times J$ matrix. To do so we need to define feature vectors u_i (rows of U) and m_j (columns of M) of size K that can approximate the rating user i would give movie j. This method reduces the number of parameters to describe X, from the number of ratings $|R|$, to $IK + KJ$, but uses real numbers in U and M instead of integers in X, which also means entries in U and M can be both positive and negative.

The intuitive way to think of this procedure is assigning each user and each item the same K identifiers, such as whether a user likes action movies and how much 'action movie' an item is. Using these identifiers we now try to match items to users in such a way that the identifiers for the user are as close as possible to the identifiers of the item. It should be noted that these identifiers do not actually exist, they are a made up mathematical construct which can be used to optimize a recommender algorithm. This is reflected in the fact that we can freely change K without first asking all the users to tell us how they like each of these K new identifiers. It has been implied, but because of its importance we will explicitly say it here, these identifiers have *not* been entered by the users for themselves or by the service for the items. We are to find the best values for these hypothetical identifiers by minimizing some error measure.

To find the optimal matrices U and M, we want to minimize the squared training error SE (which is equivalent to minimizing the RMSE) on the known elements of X. We let u_i denote the i-th row of U and m_j the j-th column of M, then the prediction for the rating that user i would give to movie j, $\hat{x}_{ij}$, is given by the dot product of these vectors:

$$\hat{x}_{ij} = u_i \cdot m_j. \tag{9}$$

The error is then trivially defined as the difference between the actual rating x_{ij} and the predicted rating $\hat{x}_{ij}$

$$e_{ij} = x_{ij} - \hat{x}_{ij} \tag{10}$$

and the aggregated squared error is

$$SE = \sum_{(i,j) \in R} e_{ij}^2, \tag{11}$$

in which we sum over all (user, item) pairs in X which have assigned to them a rating, which we define as the subgroup R. To minimize the SE we apply simple gradient descent. The gradient of e_{ij}^2 is given by:

$$\frac{\partial}{\partial u_{ik}} e_{ij}^2 = -2 * e_{ij} * m_{kj}, \quad \frac{\partial}{\partial m_{kj}} e_{ij}^2 = -2 * e_{ij} * u_{ik}, \tag{12}$$

The weights are then updated in the direction of steepest descent:

$$u'_{ik} = u_{ik} + \eta \cdot (2e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}), \quad m'_{kj} = m_{kj} + \eta \cdot (2e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}), \tag{13}$$

where λ is a regularization parameter to prevent large weights and η the so-called learning rate. Solving problems with many parameters from a sparse dataset usually overfits the data, which is why we introduce λ to attempt to avoid this. The regularization parameter will constrain the weights u_{ik} and m_{kj} and thus prevent overfitting the data. The learning rate η governs how far down the gradient we go with every step, if this value is too big we might never get close to the minimum but just jump around it, but if η is too small converging to the minimum will take too much time. The complete algorithm to optimize the matrices U and M is then:

1. Initialize U and M randomly from the standard normal distribution.
2. Loop until terminal condition is met or N iterations are reached
   (a) Iterate over each known element of X in the training set. Calculate the gradient of e_{ij}^2 according to Equation 12 and update the row of U and column of M corresponding to the current element according to Equation 13.
   (b) Calculate the RMSE on the test set.

The terminal condition is met when the RMSE on the test set does not decrease over two iterations.

3.2 Results

We adopt the following parameters from the MyMedialite website[2]: $N = 75$, $K = 10$, $\lambda = 0.05$, $\eta = 0.005$ and run the algorithm as explained in Section 3.1. We again use 5-fold cross-validation to verify the robustness of our results. In this case, this means that 20% of the ratings in the matrix X is randomly removed and put in a separate matrix X_{test}. Thus, we only loop over the remaining 80% of ratings in X_{train} when updating the matrices U and M. After this loop we calculate the RMSE on the 20% of ratings in X_{test}. We repeat this process 5 times, for 5 different splits of the data and the average final RMSE (or MAE) on the test set is then again a good estimator of the final accuracy on the entire data set.

Takács et al. (2007) state various normalization techniques, among which are subtracting the user mean or the movie mean from the ratings before training. In this report we will explore the difference between data that is not normalized and data normalized by subtracting the user mean from each rating. We choose subtracting the user mean with the idea that this would level the playing field between users that on average rate very high and users that on average rate movies quite low, independent of what they think of the movies.

The results for one fold without data normalization are given in Figure 1 for the RMSE as the error measure and in Figure 2 for the MAE as the error measure. These figures show that the error drops very quickly in the first few iterations, indicating that it is very easy to perform better than random with this algorithm, and that the gradient descent method can effectively find a local minimum. The figures also show that the final MAE is lower than the final RMSE, which makes sense since by definition the MAE is always lower than or equal to the RMSE (see Equation 7). Interestingly, in this fold the RMSE is lower on the test than on the training set, while the MAE is larger on the test than on the training set. This indicates that we make smaller mistakes on the test set, but more frequently make these mistakes, since, as explained in Section 2.1, the RMSE gives a higher weight to large errors as they are squared before they are averaged.

The results of all five folds of cross validation are shown in Table 3. As can be seen from the low standard deviation, the results did not vary substantially between folds. Furthermore, when taking the average of the 5 folds, the test error is higher than the training error, as one would expect, since the test subset contains (user, item) pairs that the algorithm has not seen before. Although the performance of the algorithm seems to be better on normalized data, the difference between normalizing the data by subtracting the user mean and not normalizing the data is negligible as Table 3 shows. This indicates that the user mean can be effectivelly learned by the matrix factorization algorithm and does not have to be subtracted from the dataset before training.

[2] `http://mymedialite.net/examples/datasets.html`

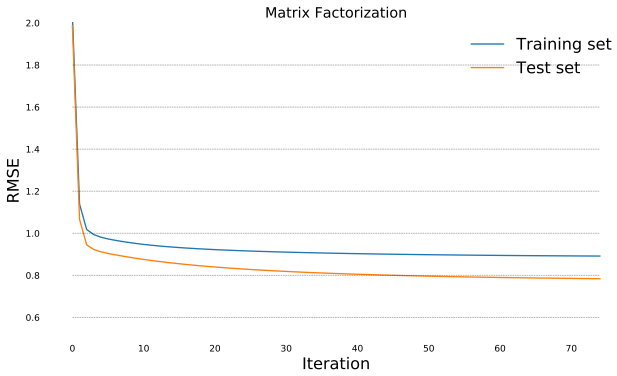

Figure 1: RMSE versus the number of iterations the matrix factorization algorithm has completed.

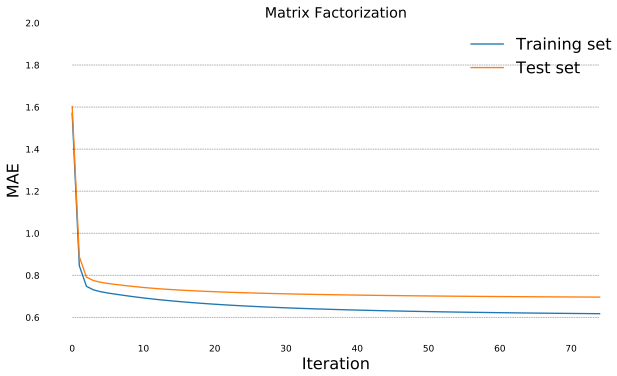

Figure 2: MAE versus the number of iterations the matrix factorization algorithm has completed.

Table 3: Results of matrix factorization with 5 fold cross validation. The mean and standard deviation of the 5 final RMSE and MAE on the training and test set are shown for both data that is normalized with the user mean and not normalized.

	Not Normalization		Normalized	
	Mean	Standard deviation	Mean	Standard deviation
RMSE (train)	0.78	0.0013	0.77	0.0005
RMSE (test)	0.89	0.0030	0.88	0.0028
MAE (train)	0.62	0.0011	0.61	0.0004
MAE (test)	0.70	0.0024	0.69	0.0020

3.3 Computational strain

This algorithm is quite complex, as it took a total of 79 minutes to complete the fivefold cross validation on a 64bit machine running Ubuntu 16.04 with 16 GB ram and the following CPU: Intel Core i7-4702MQ 2.20GHz $\times$ 8. Given that our number of users and items is actually unrealistically small, our particular implementation would most likely not work on realistic sets with billions of entries. We will nonetheless consider the scalability of our approach. To execute the matrix factorization we create an array X of size $U \times M$, which corresponds to a memory use which scales as O(UM). We need to loop over all user, item combinations we have to update entries in the matrix X with each step, this takes time O(R). Further on in the code we loop over all ratings -time O(R)- to compute the error and in the gradient descent we multiply the error with a row of length K from the matrix M, which requires time O(K). The total time then equals $O(R^2K)$ In the ideal case this could be improved upon, the creation of the matrix X is quite unnecessary and the required rating could be read from the file at the time it is required. The required memory in that case is only O(U+M). The required time then only is O(KR).

4 Alternating Least Squares

Zhou et al. (2008) describe another method for a recommendation system, the alternating least squares method. This method relies on the same principle as matrix factorization in Section 3, but the method of finding the optimal U and M differs. Formally, Zhou et al. (2008) call it 'Alternating-Least-Squares with Weighted-λ-Regularization' (ALS), and they claimed that it was one of the best pure methods out there for the Netflix challenge (see: Zhou et al., 2008). The authors define a loss function that is the following:

$$f(U, M) = \sum_{(i,j)\in R} (r_{ij} - u_i \cdot m_j)^2 + \lambda \left(\sum_i n_{ui} ||u_i||^2 + \sum_j n_{m_j} ||m_j||^2 \right) \tag{14}$$

where n_{u_i} and n_{m_j} denote the number of ratings of user i and movie j respectively. We let I_i denote the set of movies that user i has watched and I_j denote set that of users that have rated movie j. The idea of the ALS method is to alternate between solving U (and thus freezing M) and solving M (and thus freezing U). We again denote the i-th row of U as u_i and the j-th column of M as M_j. To solve for u_i we must solve

$$\frac{1}{2} \frac{\partial f}{\partial u_{ki}} = 0, \quad \forall i, k \tag{15}$$

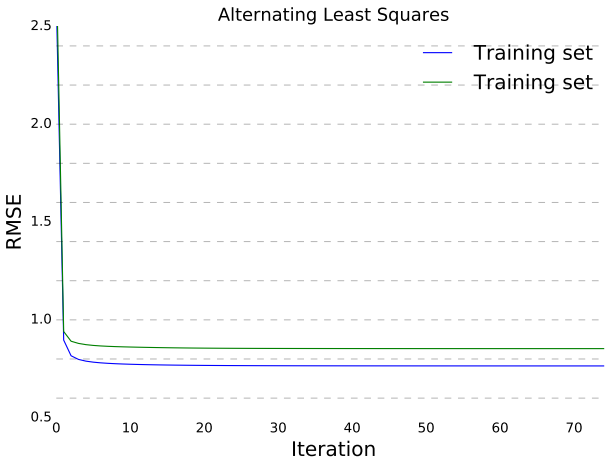

Figure 3: RMSE versus the number of iterations the ALS algorithm has completed.

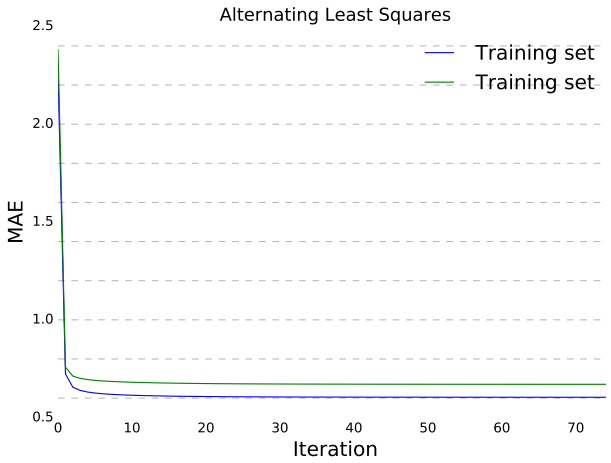

Figure 4: MAE versus the number of iterations the ALS algorithm has completed.

After some steps detailed in Zhou et al. (2008), we arrive to the following expression for solving u_i:

$$u_i = A_i^{-1} V_i, \quad \forall i \tag{16}$$

where $A_i = M_{I_i} M_{I_i}^T + \lambda n_{u_i} E$, $V_i = M_{I_i} R_{I_i}^T$ and E is the $n_f \times n_f$ identity matrix. M_{I_j} denotes the sub-matrix of M where we have selected the columns corresponding to the movies that user i has watched, and R_{I_i} denotes the ratings user i has given movies $j \in I_i$. When solving for m_j a similar result is obtained, by simply replacing i with j and U with M. To perform ALS, we initialize U and M by drawing numbers from the standard normal distribution. The method is then simply given in the following steps:

1. Fix M, solve U by looping over all users in the training set and applying Equation 16
2. Calculate the RMSE on the test set
3. Fix U, solve M by looping over all movies in the training set and applying the equivalent equation for m_j.
4. Calculate RMSE on the test set again. If this is within 10^{-4} of the RMSE in step (2) or 75 iterations are reached, the algorithm stops. Otherwise, go back to step (1).

The training and test split are again done in a 4:1 train-test ratio, and 5 fold cross-validation is again applied. The algorithm takes 45 minutes on a 64bit machine running Ubuntu 16.04 with 16 GB ram and the following CPU: Intel Core i7-4702MQ 2.20GHz × 8. The error curves are plotted in Figures 3 and 4 for a single fold of 75 iterations. As can be appreciated from these figures, the model converges fast to a local minimum. After 10 iterations the accuracy of the model does still improve above the 10^{-4} threshold, but is only improving very slightly. The results of 5 fold cross-validation are given in Table 4, from which it can be seen that the ALS method slightly outperforms the matrix factorization method, both in accuracy and robustness of the method, as both the mean and standard deviation of the RMSE and MAE are lower than in Table 3. Most noticeably, the standard deviation of the accuracy on the training sets is an order of magnitude smaller than that of the matrix factorization method, indicating that this model is less sensitive to the particular training set that is adapted from the data. Again, it seems that normalizing the data by subtracting the user mean has no significant effect on the outcomes. This indicates that the ALS algorithm can also effectively learn the user means, similar to the matrix factorization method.

Table 4: Results of the ALS algorithm with 5 fold cross validation.

	Without Normalization		With Normalization	
	Mean	Standard deviation	Mean	Standard deviation
RMSE (train)	0.76	0.0003	0.76	0.0003
RMSE (test)	0.85	0.0021	0.86	0.0024
MAE (train)	0.60	0.0003	0.61	0.0003
MAE (test)	0.67	0.0015	0.68	0.0020

The time and memory required are vaguely similar to that of matrix factorization. The matrix X is again required, memory O(UM) and time O(UKM). Then we need to loop off and on over all users or movies, time

O(U) or O(M). Next we compute the RMSE which takes a mean over the length of the predictions, which requires time O(R). In total we require time O(UKMR) and memory O(UM).

5 Conclusion

Now that we properly understand the working of recommender systems and have seen several examples of how one could build such a system, ranging from a very simply global average to the much more advanced matrix factorization and alternating least squares procedures, several conclusions can safely be drawn and more daring suggestions be made. From our results it has become clear that a huge gap lies between the accuracy of the four naïve approaches and the two more involved ones. Where the RMSE of the latter two approaches is below 0.8, with the ALS being better than the matrix factorization by a slim margin, even the best of the naïve approaches -Linear Regression- doesn't get below 0.9. The two approaches described in the second part of this report are quite remarkably close. Given that they rely on completely different methods, it is striking to see that the difference in error measure is only minimal. In future experiments, it would be interesting to see what a combination of these two methods would yield for the accuracy of a recommender system. For now we are satisfied to conclude that we can quite accurately reproduce the results reported on the MyMedialite website, this gives us confidence our implementation of the recommender systems in this report is correct. A slight difference is found with the above mentioned website for the linear regression model, this is likely due to using a different package to execute this. All together this report has shown that reasonably decent recommender systems can be build based on purely naïve approaches, they can be optimized by using more difficult mathematical constructs, but the differences are not orders of magnitude. This shows that improving recommender systems is far from trivial, which marks the whimsical nature of humanity. We might very well make perfect predictions for rational human beings, but unfortunately humans happen to posses emotions which we have not yet been able to capture in models. For now we must accept that progress can only be made by combining several advanced systems to hope that they magically model users behaviours accurately.

References

Takács, Gábor et al. (2007). "On the Gravity Recommendation System". In: *Proc. of KDD Cup Workshop at SIGKDD-07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pp. 22–30.

Zhou, Yunhong et al. (2008). "Large-Scale Parallel Collaborative Filtering for the Netflix Prize". In: *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*. AAIM '08. Shanghai, China: Springer-Verlag, pp. 337–348. ISBN: 978-3-540-68865-5. DOI: 10.1007/978-3-540-68880-8_32.