Advances in Data Mining: Implementing real-world Recommendation Systems

September 25, 2018

1 Introduction

1.1 On Recommendation Systems

Nowadays, increasing computational power has enabled researchers to obtain, access and process huge volumes of data. This development has been accurately described by Lyman and Varian[4], according to whom, the amount of collected data doubles every year. Simultaneously, as expressed by Gordon Moore, "processing speed doubles every 18 months" and analogously, the rest of the hardware follows an equivalent rate. Of course, the aforementioned "data flood" succeeds business demand and subsequently brings value. One of the characteristical domains that data are used, preprocessed and modeled on a daily basis, concerns on-line commercial stores. Customers are registered in a database where purchases are mapped to their IDs. These past purchases of theirs, as well as, knowledge from other "similar" customers enable predictive modeling on ratings, for new instances. That is, we can recommend existing products to existing customers.

Out of the daily transactions conducted and stored in the vendor's databases, data are processed accordingly, so that they are later being used to update a models' parameters. Such models are the ones yielding predictions or, by the users' perspective, recommendations. This class of models was named after recommendation or recommender systems[7] and constitutes a dynamic field of machine learning. In the following passage, we discuss the theoretical backgrounds, implement from scratch and elaborate on the results comparing several approaches.

Experimentations and benchmarking were illustrated on movies' recommendations to users. More specifically, models to be considered yield recommendations that derive from the mean of of all ratings, the mean per item and per user separately, as well as a weighted average of the two methods. Furthermore, we implement a Matrix Factorization algorithm, namely, the Gravity Recommendation System[8]. The latter was a state-of-the-art, still straightforward, approach to tackle the `Netflix Prize` [1] open competition and remained the front runner from January to May 2007. Finally, we retrain the matrix Factorization model employing the Alternating Least Squares optimization method, we explain the procedure, discuss on the differences and elaborate on the results.

[1] `https://en.wikipedia.org/wiki/Netflix_Prize#2007_Progress_Prize`

1.2 Data

As mentioned in the introductory part, models to be implemented are trained to confront the movies-to-users recommendation task. For this purpose, we make use of a `MovieLens` dataset[2]. MovieLens is a movies recommendation website, incorporating knowledge from subscripted users' profiles so as to suggest items, most likely to be highly rated by them. Data are downloadable through `GroupLens`[3] - a research lab in the Department of Computer Science and Engineering at the University of Minnesota. Dataset versions available range in size from 0.1 to 20 millions of ratings[3]. Note that, in terms of this study, we utilize the 1M dataset ; Still, models to be discussed are applicable to all datasets. Different variants of the data require different memory management strategies, so that the huge matrices emerging, are able to fit in an ordinary laptop's memory, as well as longer training procedures.

The 1M dataset, consists of 6040 registered users and 3952 movies rated on an one-to-five Likert scale. The initial format of the data provided is $< user\_id, movie\_id, rating, date >$. Even though, all users have given at least one rating, the same does not apply for movies; the number of unique ones is 3706, leaving 246 unrated items. Dates could be employed to weight ratings; Older ratings would be less important and vice-versa. Dates could be also integrated in the Matrix Factorization algorithm, as described by the authors of the paper, but findings show that the increase in number of trainable parameters is unproportional to the gains in performance. Therefore, the format that we experiment with is $< user\_id, movie\_id, rating >$. The distributional properties of the mean rating per user and the corresponding one per item are illustrated in Figure 1.

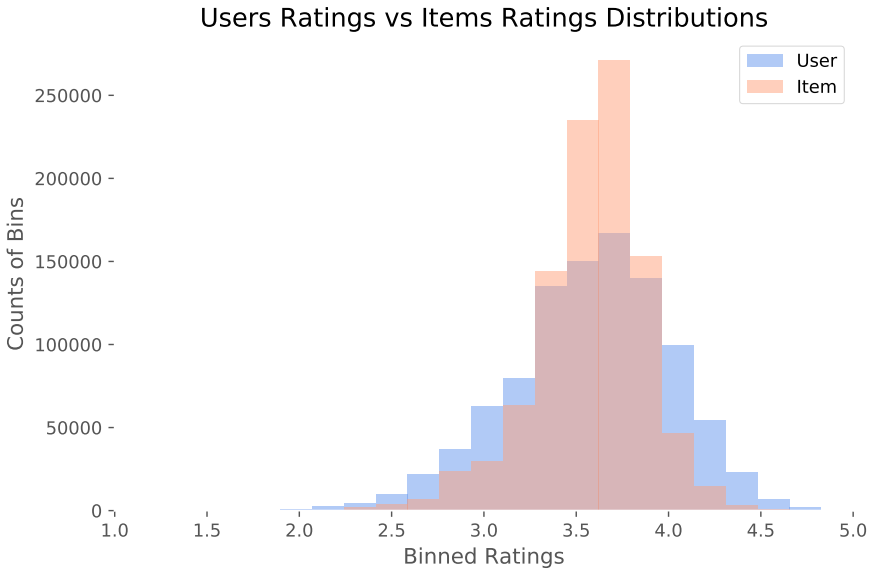

Figure 1: Average ratings distribution for user vs item

[2] `https://movielens.org/`
[3] `https://grouplens.org/datasets/movielens/`

2 Naive Approaches

2.1 Theoretical Background

Naive models are a set of easy-to-implement methods that formulate the baseline upon which comparisons with intricate models are conducted. The models that are used in this study are the (global) average of all ratings, average rating per user, average rating per item, as well as a linear regression (LR) of the true ratings on these two predicted sets, for each instance. The global average R_{global}, average rating per user R_{user} and average rating per item R_{item} are denoted as follows:

$$R_{global} = \frac{1}{n} \sum ratings \qquad \text{where } n \text{ is number of all ratings available} \tag{1}$$

$$R_{user_i} = \frac{1}{n_i} \sum user_i Ratings \qquad \text{where } n_i \text{ is number of ratings per user} \tag{2}$$

$$R_{item_j} = \frac{1}{n_j} \sum item_j Ratings \qquad \text{where } n_j \text{ is number of ratings per item} \tag{3}$$

The linear regression (or more suitably, the weighted average) model takes advantage of the estimated ratings described by the left-hand-side of eq. (2) and (3), and uses them to estimate the coefficients β_1 for the users and β_2 for the items and lastly α that serves as the intercept. The formula is denoted as follows:

$$R_{user_i - item_j} = \alpha + \beta_1 \cdot R_{user_i} + \beta_2 \cdot R_{item_j} \tag{4}$$

2.2 Implementation

The implementation of the global average method R_{global} is straightforward. For this method, only the average of all ratings needs to be computed. This global average serves as our predictions $\hat{y}_i$ for all instances i.

Furthermore, for the average rating per user R_{user_i} and average rating per item R_{item_j}, some preprocessing is required before proceeding on to the computations. If the users (or items) are not sorted by their IDs, for each of the cases, we sort them using the `argsort` function from the `numpy` library. Once users (or items) are sorted, we employ another two functions, namely, `bincount` and `cumsum` - dramatically facilitating computations. `Bincount` returns the counts of occurrence for every unique value and `cumsum` returns the cumulative sum of the counts. Technically, using these bins and cumulative sums, serves as a trick to extract the average per users (or items) more efficiently. Having obtained the intervals of indices through `bincount`, we now use them to replicate the predictions of each user (or item) rating. Since `numpy` is mostly written in `C`, the execution time for these functions is negligible. The latter serve as our predictions $\hat{y}_i$. Table 1 shows how the average rating per user (or item) is represented.

As mentioned before, the linear regression model (LR) employs both the average rating per user R_{user_i} and the average rating per item R_{item_j} and finds the optimal plane that minimizes the error. The coefficients β_1 for users, β_2 for items and α for the intercept are estimated analytically.

Note that, once the average ratings for users and items are obtained, they have to be sorted by both userIDs and then by movieIDs so that they are later mapped to the true values y. To do so, we employ a lexicographic sort, achieved by `numpy`'s `lexsort` function. Table 2 illustrates how the sorted average ratings for users and items are represented. The analytical solution for the linear model regression is denoted as follows:

user_id	R_{user}
1	4.3
1	4.3
1	4.3
2	3.5
2	3.5
...	...
n	R_{user_i}

Table 1: Average rating per user representation example

user_id	item_id	R_{user}	R_{item}
1	1	4.3	3.6
1	4	4.3	4.2
1	5	4.3	2.3
2	1	3.5	3.6
2	4	3.5	4.2
...	...	...	...
user_n	item_n	R_{user_i}	R_{item_j}

Table 2: Sorted average rating per user and item representation example

$$\hat{\beta} = (X^T X)^{-1} X^T y \qquad \text{,where } X = \begin{array}{|c|c|c|} \hline 1 & 4.3 & 3.6 \\ 1 & 4.3 & 4.2 \\ 1 & 4.3 & 2.3 \\ 1 & 3.5 & 3.6 \\ 1 & 3.5 & 4.2 \\ \ldots & \ldots & \ldots \\ 1 & R_{user_i} & R_{item_j} \\ \hline \end{array}$$

To explain the analytical algebraic solution further, X is a (design) matrix of shape 1000209 rows by 3 columns. The first column contains ones; these are needed to estimate the intercept α, while the second and third columns contain the average ratings per user R_{user} and average ratings per item R_{item} respectively for each one of the instances. On the other hand, y is a vector of length 1000209 that contains the actual ratings from the dataset. Note that $(X^T X)$ is non-singular, which means that its determinant is nonzero and therefore, it can be inverted. Dealing with a singular matrix would require the estimation of the coefficients to be conducted iteratively (numerically). Additionally, $\hat{\beta}$ is a vector of length 3 that contains the estimated values for α, β_1 and β_2 respectively which, in statistical terms, minimizes the residual sum of squares (RSS)[2]. Finally, before computing the errors as $\epsilon_i = \hat{y}_i - y_i$, we sort y_i lexicographically to impose the same ordering on the ratings as the one imposed on the predictions $\hat{y}_i$.

2.3 Results

To evaluate performance on unseen data, we employ the 5-fold cross-validation method measuring the Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE). RMSE and MAE are formulated as follows:

$$RMSE = \sqrt{\frac{\sum \epsilon_i^2}{n}} \quad MAE = \frac{|\sum \epsilon_i^2|}{n}$$

Subsequently, we shuffle using seed value of 17 and partition the whole dataset in 5 folds; We train the model on four of them and evaluate on the remaining one - the test set. The RMSE and MAE averaged over all the folds are estimates of its performance. For each of the methods we obtain the predictions and evaluate the RMSE and MAE respective for each of the folds. The results of each of the methods are reported in Table 3 along with the time elapsed.

Looking at the observed results, the weighted Average method (LR) was the best in terms of obtaining the lowest RMSE and MAE averaged over the 5 folds of cross-validation performed. The

	Time (seconds)			Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
R_{global}	1.46	RMSE	Train	1.1173	1.117	1.1175	1.1175	1.1167	1.1171
			Test	1.1162	1.1175	1.1175	1.1157	1.1186	1.1171
		MAE	Train	0.934	0.9339	0.9338	0.9342	0.9334	0.9339
			Test	0.9334	0.9335	0.9342	0.9325	0.9355	0.9338
R_{user}	3.38	RMSE	Train	1.0279	1.0278	1.0277	1.0278	1.0272	1.0277
			Test	1.0155	1.0157	1.0156	1.0157	1.0179	1.0161
		MAE	Train	0.8224	0.8229	0.823	0.823	0.8224	0.8227
			Test	0.8134	0.8117	0.8116	0.8111	0.814	0.8124
R_{item}	8.9	RMSE	Train	0.9746	0.9742	0.9745	0.9748	0.9738	0.9744
			Test	0.9675	0.9688	0.9677	0.9665	0.9703	0.9682
		MAE	Train	0.7784	0.7785	0.7784	0.7792	0.7781	0.7785
			Test	0.7736	0.7736	0.7733	0.7708	0.7749	0.7732
$R_{user_i - item_j}$	13.32	RMSE	Train	0.9149	0.9145	0.9148	0.915	0.9141	0.9147
			Test	0.9004	0.9016	0.9	0.8999	0.903	0.901
		MAE	Train	0.725	0.7248	0.7251	0.7253	0.7244	0.7249
			Test	0.7136	0.7141	0.7127	0.7118	0.7149	0.7134

Table 3: Naive Approaches Results

RMSE and MAE on the train sets were 0.9147 and 0.7249 respectively and the RMSE and MAE on the test set were 0.901 and 0.7134 respectively.

The average rating per item method R_{item} was the runner-up with 0.9744 and 0.7785 for the RMSE and MAE on the train set respectively. In addition, the RMSE and MAE values on the test set were 0.9682 and 0.7732. Third, the average rating per user method R_{user} performed slightly worse, yielding an RMSE of 1.0277 and a MAE of 0.8227 on the train set whereas corresponding values on the test set were 1.0161 and 0.8124. At this point one remark should be made; As Figure 1 illustrates, variance of users' means is larger (0.436) than items' means one (0.298). Subsequently, predictions based on items would yield lower error, as was finally obtained.

Last, the global average method R_{global} yielded approximately the same RMSE for both train and test sets with value 1.1171 whereas respective MAE values were 0.9339 and 0.9338. Moreover, the time needed for each of the methods reflects the complexity in terms of operations performed to estimate parameters and output predictions.

3 Matrix Factorization via Stochastic Gradient Descent

3.1 Theoretical Background

Matrix factorization or Matrix Decomposition is the concept of factorizing a matrix into the product of other matrices[6]. Singular value decomposition (SVD) is one such technique. The initial popularity matrix X is decomposed to the matrices U, Σ and M such that:

$$X = U\Sigma M$$

where $U \in R^{I,min(I,J)}$, $M \in R^{min(I,J),J}$ and $\Sigma \in R^{min(I,J),min(I,J)}$, with U, M being orthogonal and Σ being a square matrix with non-zero elements only across the main diagonal. The latter are named

after singular values. Retaining just a number k of them, and forcing the rest to zeros, we achieve a lower-rank approximation of the initial matrix X. That is, matrix X is now being approximated by the product of U, Σ' and M where Σ' is the lower-rank version of Σ .The former is now a matrix retaining k components of the initial matrix. The number of singular values preserved corresponds to the number of components the approximation retains.

The concept behind the employment of SVD for recommendation systems is that we can learn a training set of N ratings to J movies from I users. Decomposing the training set and retaining the k-th components, we "learn" the affinities between users and movies by means of weights. Weights are the matrices U and M having incorporated matrix Σ' as following:

$$X = (U\Sigma')M = (U\sqrt{\Sigma'})(\sqrt{\Sigma'}M)$$

For ease of use, let us retain the names U and M for the new matrices $U \in R^{I,k}$ and $M \in R^{k,J}$. Each row-vector of U is a k-dimensional mapping of the i-th user to the k-th component. Respectively, each column-vector of M is a mapping of the k-th component to the j-th movie. The product of the two matrices unifies users and movies into the approximation matrix $\hat{X}$ which contains the approximations (predictions) of the initial matrix X. Consequently, a model that has learned the preferences is one that minimizes the approximation error, or in other words, the loss function $\parallel X - \hat{X} \parallel$. Mathematically, these are denoted as follows:

$$\hat{x}_{ij} = \sum_{k=1}^{K} u_{ik} m_{kj} \tag{5}$$

$$\epsilon_{ij} = \hat{x}_{ij} - x_{ij} \tag{6}$$

$$SE = \sum_{ij} \epsilon_{ij}^2 \tag{7}$$

SE is the approximation error that we wish to minimize. This is proportional to minimizing the Root Mean squared Error (RMSE). This minimization is conducted numerically. Optimization routine employed for this task is the first-order iterative Gradient Descent. Weights U, M are randomly initialized so that the optimization does not get stuck on a local minimum. We subtract the approximation $\hat{X} = UM$ from the popularity matrix X so that the errors are obtained. Then, we find the derivatives of the errors with respect to both u_{ik} and m_{kj} weights. This is formulated as follows:

$$\frac{\partial}{\partial u_{ik}} \epsilon_{ij}^2 = -2\epsilon_{ij} m_{kj} \qquad \frac{\partial}{\partial m_{kj}} \epsilon_{ij}^2 = -2\epsilon_{ij} u_{ik}$$

To preserve weights inflation we impose a penalty λ on the structure of the update rule:

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

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

Upon convergence, the obtained weights U, M are the optimal parameters given that the model is not overfitting. More on this are to be discussed in the next section. Once the model has learned i-th user's preferences, it can recommend the j-th movie, for which the predicted value $\hat{x}_{ij}$ is the maximum for this fixed i-th user.

3.2 Implementation

Implementation of the above technique is quite straightforward. Note that, for this task, the popularity matrix X $\in R^{6040,3952}$, that is, we incorporated movies for which there are no ratings by any users. This does not affect the final outcome as a prediction for an new instance would be eq.(5). Concerning an unrated movie, the weights learned during training are $m_{kj} = 0$ for this zero j-th column of matrix M. Subsequently such an instance will be coerced to zero. Equivalently, the error eq.(6) and its gradients are zero, not having an impact on either the training or the evaluation processes.

Again, cross validation was employed to measure performance. Note that, we do not loop through all the elements of the popularity matrix; This would be inefficient when working with huge sparse matrices. Instead, we loop through the indices of elements that have non-zero records. Accordingly, 1M ratings were split to 5 folds - for each fold, the procedure followed is described in Algorithm 1.

Algorithm 1 Matrix Factorization with Gradient Descent

procedure CROSS VALIDATION(k-th fold)
  Randomly initialize weights `U,M` and set λ, η
  for `75 reps` **do**
    for `each rating` x_{ij} `on the train` **do**
      `find approximation` $\hat{x}_{ij}$`, compute` ϵ_{ij}
      `compute derivatives of errors w.r.t.` u_{ik}, m_{kj}
      `update weights` u'_{ik}, m'_{kj}
  Evaluate `RMSE, MAE` on train and test

3.3 Results

The Matrix Factorization model was trained many times, under different parameters settings. Matrices U, M were randomly initialized and seed value 1992 was used. Number of iterations is set to 75 - many experiments have indicated the number is sufficient for minimizing the loss function while preventing overfitting. The latter is also addressed by learning a small number of k features. Intuitively, the more the features learned from the training set, the less generalizable the model is. Each recommendation system trained with Matrix Factorization sets the k number of features depending on its dimensions.

Four experiments for the learning parameter were conducted, using different values for η, namely, 0.001, 0.005, 0.01, 0.05 while fixing $\lambda = 0.05$. Convergence rates are illustrated in Figure 2. Minimization of the loss function is achieved faster for $\eta = 0.005$. Keeping η fixed, different experimentations were performed, with k = 20 and $\lambda = 0.05$, as well as k = 20 and no penalty λ. Table 4 illustrates performance of the two models in the train and test set ; both of them learn details of the training set and cannot generalize so that they are able to make credible recommendations. Note that, in contrast to k = 10, k = 20 creates matrices of double size $U \in R^{6040,20}$ and $M \in R^{20,3952}$ - number of parameters doubles equivalently. Finally, optimal settings turned out to be $\lambda = 0.05$, $\eta = 0.005$, $k = 10$, $nrIters = 75$ yielding an RMSE of 0.877 and a MAE of 0.687 on the test set.

3.4 Algorithmic Complexity

Algorithmic Complexity applies to both time and memory management. Complexity is a function of n, the number of steps, and denoted as T(n). Then, T(n) is measured in terms of the Big-O

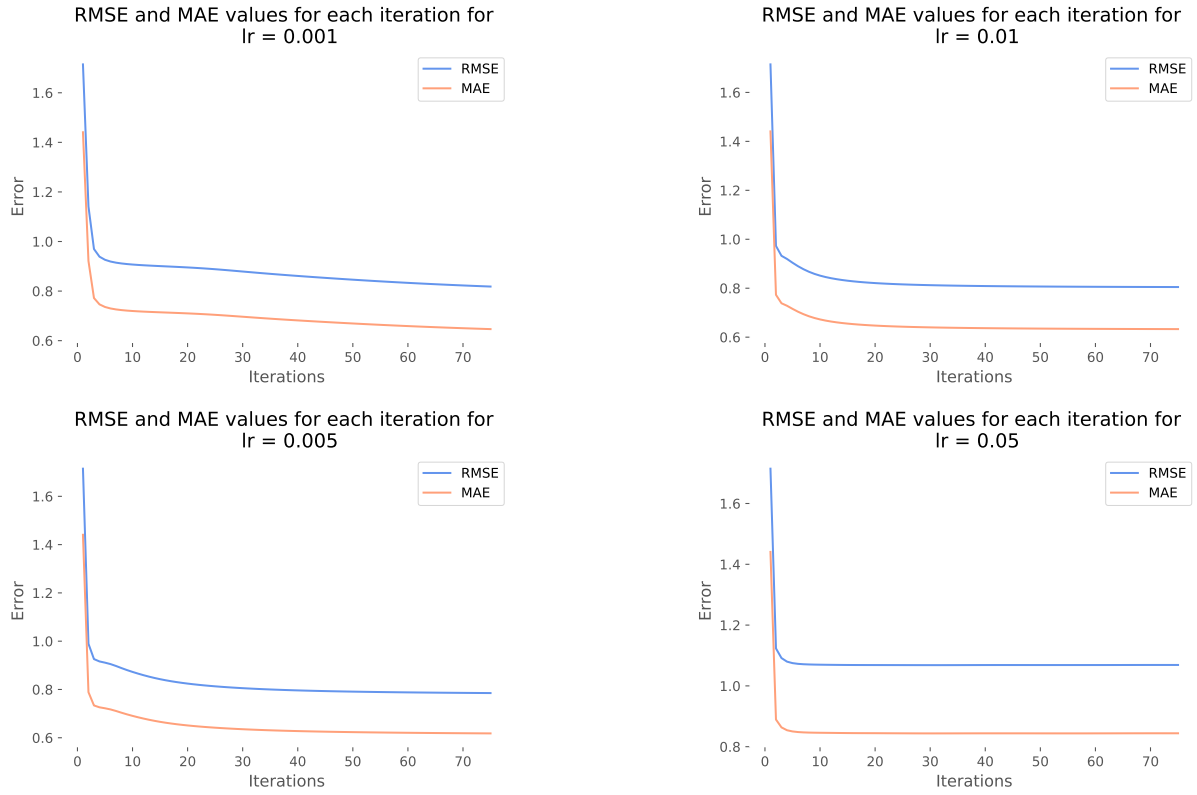

Figure 2: RMSE and MAE metric monitored across iterations

notation[5][1]. Note that, Big-O notation aims to describe complexity regardless of the implementation, thus, describes complexity asymptotically.

In our case, given that our records is N (1M), number of iterations is n (75) and constants are mapped to zero (such that O(c) + O(N) = O(N)), Big-O notation for running time is O(N). For n steps, that would be O(nNc) but as explained earlier, n and constants vanish from the formula as N becomes bigger. Similarly, given that I' and J' denote the number of unique users, big-O notation for memory requirements is O(I',J'). This describes internally the number of multiplications and additions required element-wisely.

4 Matrix Factorization via Alternating Least Squares

4.1 Theoretical Background

In this section, we describe a Matrix Factorization algorithm trained via Alternating Least Squares[9]. That is, a second-order iterative procedure employed for optimization problems. Again, as described in the previous section, weights matrices U, M are determined, so as they yield a k-rank approximation of the initial data matrix X. Now, while a criteria holds, we update weights U while fixing M and vice-versa, interchangeably.

Matrices U, M are updated vector-wisely and rules are denoted in the paper authors' notation:

$$u_i = A_i^{-1} V_i, \qquad m_j = A_j^{-1} V_j$$

Where, $A_i = M_{I_i} M_{i_i^T} + \lambda n_{u_i} E$, $\quad V_i = M_{I_i} R^T(i, I_i)$, E is the identity matrix and M_{I_i} is a sub-matrix

η	k	λ	Average time fold (minutes)			Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
0.001	10	0.05	26.3	RMSE	Train	0.816	0.819	0.816	0.814	0.818	0.817
					Test	0.876	0.879	0.875	0.874	0.877	0.876
				MAE	Train	0.645	0.647	0.644	0.643	0.647	0.645
					Test	0.689	0.692	0.689	0.687	0.690	0.689
0.01			38.4	RMSE	Train	0.806	0.806	0.806	0.805	0.805	0.806
					Test	0.897	0.901	0.898	0.898	0.899	0.899
				MAE	Train	0.634	0.634	0.634	0.633	0.633	0.634
					Test	0.705	0.707	0.705	0.704	0.705	0.705
0.005			29.2	RMSE	Train	0.786	0.786	0.787	0.785	0.785	0.786
					Test	0.875	0.879	0.876	0.877	0.876	0.877
				MAE	Train	0.618	0.619	0.619	0.618	0.618	0.618
					Test	0.686	0.688	0.687	0.687	0.686	0.687
0.05			34.6	RMSE	Train	1.069	1.068	1.069	1.068	1.069	1.069
					Test	1.181	1.177	1.176	1.175	1.186	1.179
				MAE	Train	0.844	0.843	0.844	0.844	0.844	0.844
					Test	0.937	0.934	0.935	0.933	0.942	0.936
0.005	20		33.3	RMSE	Train	0.722	0.721	0.722	0.721	0.721	0.721
					Test	0.895	0.893	0.895	0.894	0.896	0.895
				MAE	Train	0.567	0.566	0.567	0.566	0.566	0.566
					Test	0.699	0.697	0.699	0.698	0.698	0.698
		None	34.8	RMSE	Train	0.749	0.748	0.749	0.749	0.747	0.748
					Test	0.910	0.907	0.908	0.909	0.908	0.908
				MAE	Train	0.588	0.588	0.588	0.588	0.587	0.588
					Test	0.709	0.707	0.709	0.708	0.707	0.708
ALS	10	0.05	2.04	RMSE	Train	0.765	0.764	0.764	0.763	0.764	0.764
					Test	0.852	0.853	0.851	0.853	0.852	0.852
				MAE	Train	0.603	0.603	0.603	0.602	0.603	0.603
					Test	0.669	0.67	0.669	0.67	0.669	0.670

Table 4: Table of Results: Matrix Factorization
Note the optimal model having its RMSE scores denoted with blue color. Red color is used to highlight the overfitting models. Both of them perform better on the train set, but are outperformed on unseen (test) data. Blue color applies for the GD and ALS MF models, both with $\lambda = 0.05$ and $k = 10$.

of M referring to the movies user i rated. Identically, A_j, V_j are defined. λ stands for the penalty imposed on the weights.

4.2 Implementation

Once notation is clear, the implementation of the algorithm is straightforward. Still, one extra step needs to be taken. Note that, there are some never rated movies (246). When these movies are mapped to their k-dimensional vectors in the training phase, vectors cannot be updated because their A_j^{-1} matrix is a zero scalar, that is, a singular one, and thus cannot be inverted. At this point, we come up with an ad-hoc solution; We create an escape rule, according to which if the matrix is

non-singular we update according to the rule, else we impute the m_j k-dimensional vector of the M matrix with its row means. So, instead of filling with random noise - which is the baseline anyway - we decided to employ the information we have for the rest of the movies on this k-dimensional space. This, turned out to lift performance considerably.

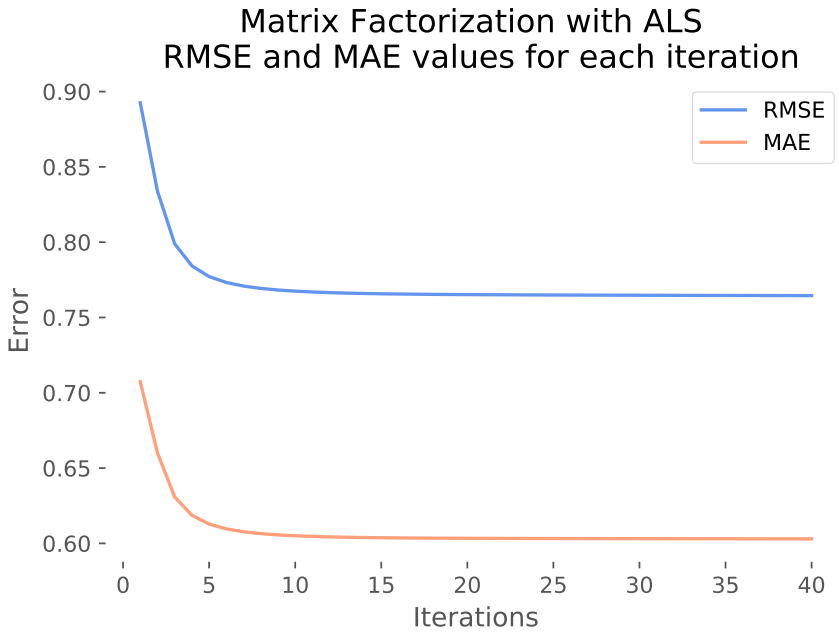

Figure 3: RMSE & MAE on the y-axes versus iterations on the x-axes.

4.3 Results

Using a stopping difference of 0.0001, as proposed by the authors of the paper, the algorithm converges and evaluates each fold in two minutes - that is, a huge increase in efficiency. The convergence rates are illustrated on 3 Due to the training facilitation, we are able to further experiment with the algorithm and reach RMSE values of 0.764 0.852 on the training and test set respectively. Values for the MAE metric, and separate folds performance, are depicted on Table 4, in the last entry. The ALS-trained Matrix Factorization algorithm turned to be, overall, the best model in terms of performance and decreased computational burden.

5 Conclusion

In this paper, we develop different methods to build a recommendation system for the 1M MovieLens dataset. All these methods were evaluated using the Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE) over 5 folds of cross-validation. The average value of all folds was the measure of performance, in this context.

First we implemented four naive methods. The first approach was based on the global average of all ratings. The second and the third ones were based on the average rating of each movie and the average rating of each user. Also, a weighted average of the two was implemented to estimate the regression weights that minimize the error. The latter yielded the best results on the test set, yielding RMSE and MAE values of 0.901 and 0.7134. RMSE values for user and item average ratings are 1.016 and 0.968 respectively, whereas, for the global mean it is 1.117.

Following that, we moved on to implement the Matrix Factorization algorithm trained with the Gradient Descent. We experimented with different parameters settings such as learning rate η of 0.001, 0.005, 0.01 and 0.05 with a fixed penalty $\lambda = 0.05$ and features $k = 10$. Furthermore, We showed that, by increasing the number of features k or removing the penalty we overfit the training set. The optimal model recorded was the one with a reported value of RMSE 0.877 and MAE 0.687 on the test set, trained with $\lambda = 0.05$, $\eta = 0.005$, $k = 10$, $nrIters = 75$.

Finally, the Matrix Factorization model was re-trained using the ALS optimization method, achieving huge decline in computational costs, simultaneously with even an increase in performance. The optimal parameters settings were for k=10 and $\lambda = 0.05$ yielding RMSE and MAE values of 0.852 and 0.670 correspondingly, on the test set.

References

[1] P Danziger. Big o notation. *Source internet: http://www. scs. ryerson. ca/˜ mth110/Handouts/PD/bigO. pdf, Retrieve: April*, 2010.

[2] John Fox. *Applied regression analysis, linear models, and related methods.* Sage Publications, Inc, 1997.

[3] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

[4] Martin Hilbert. A review of large-scale 'how much information'inventories: variations, achievements and challenges. 2015.

[5] Donald E Knuth. Big omicron and big omega and big theta. *ACM Sigact News*, 8(2):18–24, 1976.

[6] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[7] Francesco Ricci. Liorrokach, and brachashapira." introduction to recommender systems handbook, 2011.

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

[9] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *International Conference on Algorithmic Applications in Management*, pages 337–348. Springer, 2008.