Using Matrix Factorization Algorithms For Constructing A Movie Recommendation System Based On Collaborative Filtering

Subject: Mathematics for Automatic Learning - Project 1

André Roque 86694

Instituto Superior Técnico - Universidade de Lisboa

Abstract

Using a dataset from a movies ranking database, two different matrices were produced using different approaches for the missing values (replacing them by zeros and by the respective column mean value) and three matrix factorization methods (using Singular Value Decomposition, Alternated Least Squares and Non Negative Matrix Factorization techniques) were applied in order to predict the ratings that users would give to unseen movies. The aim was to evaluate the reconstruction errors of these algorithms, to compare the precision of each prediction produced and to get some insights of the computation cost of them as well. It was made also a latent space projection for each of the algorithms.

The algorithm using the Singular Value Decomposition significantly outperformed the other two, by both means of precision of the prediction and the computational cost. Even for some low rank values, the predictions computed using the mean value for the imputation of the missing values were relatively precise. The latent space projection showed some aspects of the ratings distribution.

Keywords: Recomendation Systems, Collaborative Filtering, Non Negative Matrix Factorization, Singular Value Decomposition, Alternated Least Squares, Dimensionality Reduction, Latent Space Projection.

1 Introduction

In modern society, we are often confronted with the term recommendation system. In a general way, a recommendation system is a information filtering system that uses data to predict the "preference" a user would give to an item [1]. These recommendation systems can be created based on two main ideas[2]:

1. The **collaborative filtering** approach that, for each user, creates a profile that describes his preferences using the information of similar users, by analysing relationships between users and similarities among the products
2. The **content filtering** approach that uses the past behaviour information and demographics data of an user to create a profile and predict his choices

This work is focused on the first approach.
In various applications, like in Movie Ratings Systems, many ratings are not available, since for that to be accomplished every user would need to evaluate every movie that is available on that system. For building a recommendation system for that case, the system has to infer some of the user preferences; that is, it is necessary for the recommendation system to build a model that fits the user choices. These models are typically built in such a way that it minimizes some cost function. Since this minimization processes are usually a computationally expensive procedures, relying on a huge amount of data, there is a trade-off between the precision of those models and the computation cost they require [3]. The objective of this work is to analyse this trade off for three different matrix factorization algorithms.
Given a dataset containing information about ratings that users gave to a set of movies, 3 matrix factorization methods were applied in order to predict the ratings that users would give to unseen movies. The aim was to evaluate the reconstruction errors of these algorithms, to compare the precision of each prediction produced and to get some insights of the computation cost of them as well.

2 Methods

2.1 The dataset

The dataset used was one of the latest MovieLens datasets, a set of datasets that resulted from the interaction between users and the MovieLens online recommender system, first released in 1998, that has seen over the past years a huge increase on that amount of users and evaluations made by them. [4]. This particular dataset is the movie-lens-small-latest-dataset , and it contains 100836 ratings (given on a scale from 0.5 to 5, with 0.5 and 5 being respectively the worst and the best ratings a user could give to a movie accordingly to their preferences) made by 610 users over 9724 different movies. The average rating is 3.5 with a standard deviation of 1.04.

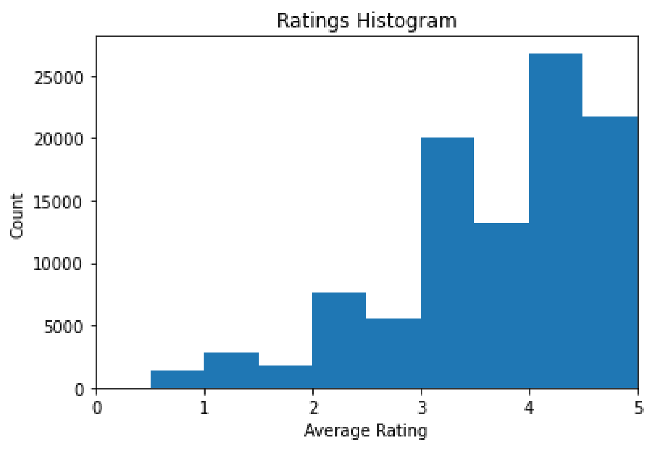

Figure 1: Distribution of the ratings in the dataset used

The quartiles of the ratings are 3.0, 3.5 and 4.

2.2 SVD - Singular Value Decomposition

Given a Matrix M, with dimensions $m \times n$ the SVD of M is

$$M = U\Sigma V \tag{1}$$

where the columns of U and V are the left and right *singular vectors* of M, and Σ is a diagonal matrix $diag(\sigma_1, .., \sigma_r, 0, ..., 0)$ where each σ_i is a *singular value* of A. That is, U and V are $m \times m$ and $n \times n$ orthonormal matrices that are bases for R^m and for R^n and the first r diagonal entries of Σ, with r the rank of M, are the eigenvalues of the matrix M. [5]

Let M_k be the the matrix defined by

$$M_k = \sigma_1 u_1 v_1^t + ... + \sigma_k u_k v_k^t \tag{2}$$

where the $\{u_i\}$ and $\{v_i\}$ are the column vectors of the matrices U and V. Then, according to the Eckart-Young theorem, the matrix M_k is the best k rank matrix approximation of the matrix M, for any matrix norm. [6]

The algorithm implementation essentially copies this idea, calculating the SVD decomposition for each matrix and truncates this decomposition as it is suggested above according to a value of k provided.

For the SVD decomposition of the matrices it was used the function *linalg.svd* from the python library *NumPy* [7].

2.3 NMF - Non Negative Matrix Factorization algorithm

Let V be the $N \times K$ data matrix, with column vectors v^a, with $a = 1, .., K$ and dimension n, and a h^a the correponding coefficient vectors. Then the linear approximation of v^a is given by

$$v^a \approx \sum_{i=1}^{M} w_i h_i^a = Wh^a \tag{3}$$

where W is a $n \times m$ matrix with the basis vectors w_i as columns a h^a is the coefficient vector.

If H is matrix formed by the column vectors, we can rewrite the previous equation as

$$V \approx WH \tag{4}$$

The optimal choice for matrices W and H is to define them as non negative matrices that minimize the reconstruction error between V and WH. In order to obtain such matrices, it was used the algorithm presented below, described in [8] and in [9] without applying the referred sparseness constraints.

Algorithm 1: NMF without sparseness constraints

```
1. Initialize matrices $W$ and $H$ with random positive numbers
2. Iterate until convergence:
    (a) $W := W \otimes ((VH^t) \oslash (WHH^t))$
    (b) $H := H \otimes ((W^tV) \oslash (W^tWH))$
3. Output $WH$
```

where $\otimes$ and $\oslash$ denotes the elementwise multiplication and division. Using as cost function the Euclidean Distance, after each iteration the algorithm converges to the optimal solution [9] . In the presented implementation of the algorithm, the input parameters are a matrix M, a value k for the pretended rank, a number of maximum iterations n, a value e for minimum norm of difference between the matrices after each iteration, and the norm that is applied for calculating the difference. Being $V_i = W_iH_i$ the matrix obtained after the ith iteration, the algorithm outputs V_{i+1} if the following condition is satisfied

$$||V_i - V_{i+1}||_{\text{norm}} < e \tag{5}$$

If not, the algorithm performs n iterations and returns V_n. The initialization of the matrices W and H is made with random integers in the range from 1 to 5.

2.4 ALS - Alternating Least Squares

Let X be a $m \times n$ matrix with the observed entries indexed by Ω and let the projection $P_\Omega(X)$ be the $m \times n$ matrix with the values of X and the missing entries equal to 0. For completing the entries of X, it was proposed by Srebro et al. [10] the following minimization problem

$$\min_{A,B} F(A,B) = \frac{1}{2}||P_\Omega(X - AB^t)||_F^2 + \frac{\lambda}{2}(||A||_F^2 + ||B||_F^2) \tag{6}$$

where A and B are $m \times r$ and $n \times r$ matrices, λ is a parameter such the smallest of the singular values is less than λ and $||.||_{Fro}$ is the Frobenius norm. The solution of the ALS algorithm presented pseudo code below converges to a solution of this problem as it is shown in [11] .

Algorithm 2: ALS

```
1. Initialize matrices $A$ and $B$ with random numbers
2. Iterate until convergence:
    (a) $B := (A^tA)^{-1}A^tX$
    (b) $A := XB^t(BB^t)^{-1}$
3. Output $AB$
```

Just like in the NMF algorithm, in the implementation of the ALS algorithm the input parameters are the same and are subjected to the same conditions. Also, the initialization of the matrices A and B is made with random integers in the range from 1 to 5.

2.5 Performance Evaluation

From the ratings dataset, two matrices were built, R and $R2$. In those matrices, the entries r_{ij} represents the rating that the user j gave to the movie i. The only difference between these 2 matrices is the way that the missing values were handled: each missing value on R was replaced by a 0, producing a sparse matrix, and on $R2$ each missing value was replaced by the mean rating that every user gave to a certain movie.

In order to evaluate the error produced by each factorization algorithm, after applying each factorization algorithm, it was produced a matrix with rank k, where k is defined as an input parameter of each algorithm ,and the reconstruction error was computed for each value k[1] using the Frobenius norm (that is, the extension of the euclidean norm to $\mathbb{R}^{n\times n}$) on the matrix obtained from the difference between the original matrix and the matrix produced by each algorithm as described by the following formula

$$E(M, M_a) = ||M - M_a||_{Fro} \tag{7}$$

In order to perform a cross validation of the results, from each of those matrices, two other matrices were created, one with 90% of the known ratings - the train matrix- and the other with the remaining 10% - the test matrix.

For the cross validation error, it was computed, for each k, the Sum of Squared Errors (SSE) and the Standard Error of Estimate (SEE), between the original M ratings on each of the test matrices and the respective "predicted" values of the k rank matrices that resulted from applying each algorithm to the two train matrices, as described by the formulas below:

$$SSE = \sum_{i=1}^{M} (r_{i_{predicted}} - r_{i_{original}})^2 \tag{8}$$

$$SEE = \sqrt{\frac{\text{SSE}}{\text{M}}} \tag{9}$$

[1]It was used every value from 1 to 50 and afterwards it was used a step of 10 until k=300

2.6 Latent Space

The idea behind matrix factorization is to represent users and items in a lower dimensional latent space. Matrix factorization learn lower rank representations of users and items in a joint latent space of a much lower dimension than the original the number of users or items. The similarity between users preferences can be obtained through a computation of their latent factor space representation. [12].

For each algorithm applied, is has been made an analysis of the latent space associated with the first 100 movies for k = 2 and the respective points were plotted.

3 Results and Discussion

3.1 Algorithms Results

The results of the three algorithms are presented below.

The SVD algorithm computation time was largely lower comparing with the other 2 algorithms. This was mainly to the computational complexity of the algorithm, since the most expensive step was to calculate the SVD decompositions of the matrices R and R2 and this computation was only needed to be done once for each matrix.

For the ALS algorithm, it was used as parameters n=10 for the maximum number of iterates and e=10. In order for the computation of each matrix with rank k to be done in a accepted amount of time, the stopping error had to be set high; nonetheless, since the difference $||V_i - V_{i+1}||_{norm}$ between the matrices produced by two consecutive iterations was verified to be strictly decreasing, and so the results would not be as far from the ones a lower value for e would produce as one may thought. The maximum number of iterates was never reached, what confirmed the value for n to be a good choice, for this value of e. In each iteration of the matrices A and B, it was used an approximation of the real value to 3 decimals, since it greatly reduced the effort of the computations and experimentally did not considerably affect the results.

In what regards to the NMF algorithm, the parameters applied were e=1 and n =50. It was the most computationally expensive algorithm and for some values of k, the maximum number of iterations was reached, meaning that the error for those specific iterations could be large.

For each algorithm, the reconstruction error can be seen in figures 2 ,3 and 4. The reconstruction error of the R matrix was similar for every algorithm, but for the R2 matrix the SVD algorithm slightly outperformed the other two. The difference between the reconstruction error seen on the curves for the matrices R and R2 can be explained by the fact that the matrix R being a sparse matrix, and for that reason lower rank matrices would approximate best the R matrix, as opposed to what is shown for the matrix R2, being the best approximations the one with the higher rank (that is, the ones wich contained more "information" about the original matrix.

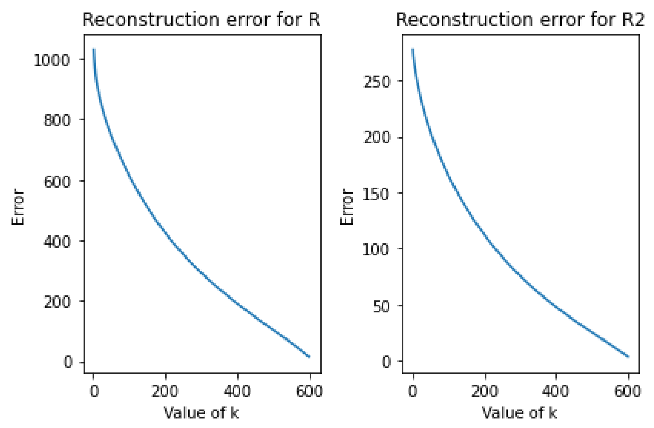

Figure 2: Reconstruction Error for the SVD algorithm using Frobenius norm

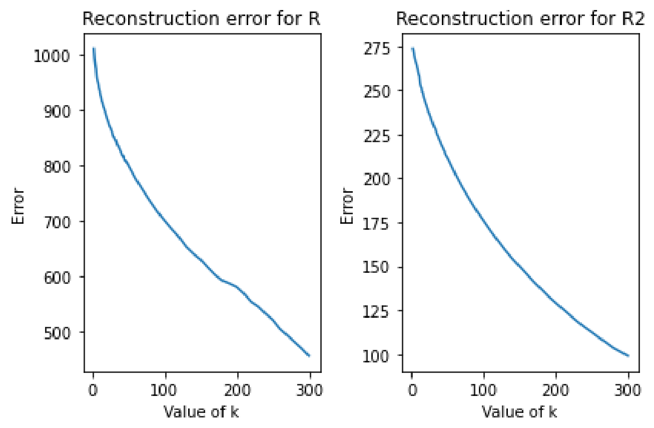

Figure 3: Reconstruction Error for the ALS algorithm using Frobenius norm

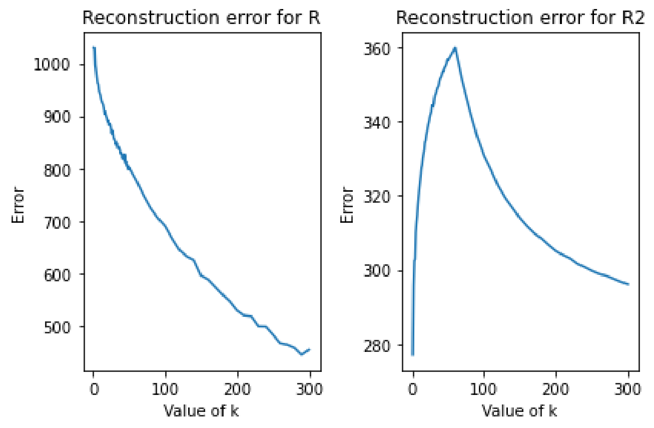

Figure 4: Reconstruction Error for the NMF algorithm using Frobenius norm

In what regards the cross validation results of the performance of the algorithms is shown in figures 5, 6 and 7. Despite the similarity between the SSE and SEE curves of each algorithm, the SVD algorithm clearly outperformed the other two algorithms predicting the test values using the R2 matrix. This result was already expected due to Eckart-Young theorem [6] that says that the approximation produced by the SVD decomposition is the best possible approximation of rank k, which is according to this work results.

The imputation of the missing values using the mean value (in the R2 matrix) instead of using zeros (like in the R matrix) showed to be a more precise heuristic, given the huge difference of the SSE and SEE obtained with any of the algorithms. Besides the mean value be a closer approximation of the real values, the use of zeros would produce some.

On the other hand, the computations made with the R matrix showed to be much more faster, suggesting a trade off accuracy vs computational cost between these different two approaches. It also should be noted that for the for lower values of k, the predictions by made each algorithm for the R matrix were more precise. This is mainly due to the fact that lower rank approximations of R would contain also a lower amount of zeros, given the sparsity of this matrix; for that reason, the approximations produced by the algorithms would contain also less zeros, and therefor the results be less biased of wrong information.

It should also be noticed that, even for low values of k, the SEE values obtained with the SVD algorithm using the R2 were under 1, a result that suggests that this algorithm can in fact provide some predictions and with low values of k can be used by recommendation systems at some extent, with reduced computational cost.

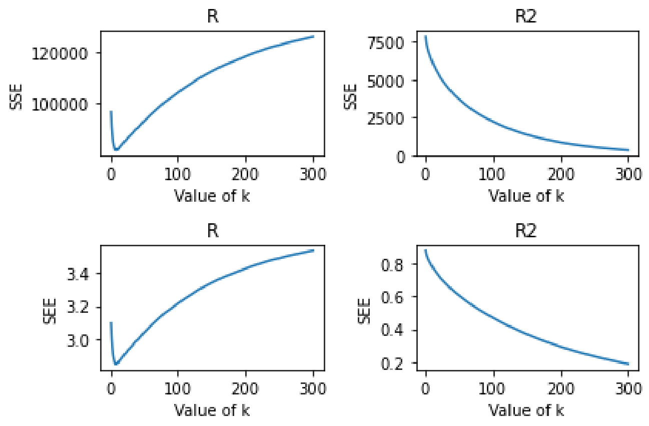

Figure 5: SSE and SEE for the SVD algorithm for the matrices R and R2

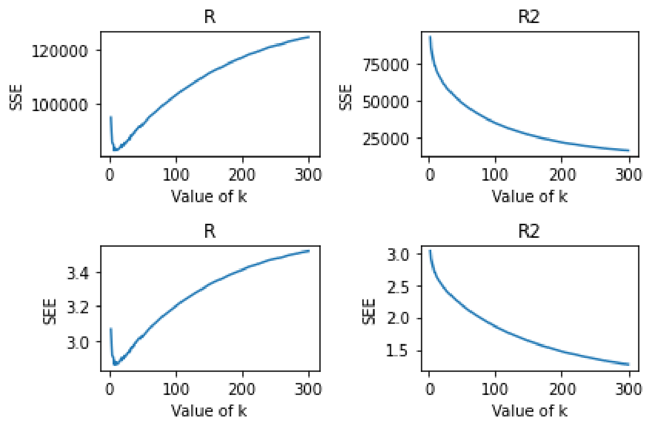

Figure 6: SSE and SEE for the ALS algorithm for the matrices R and R2

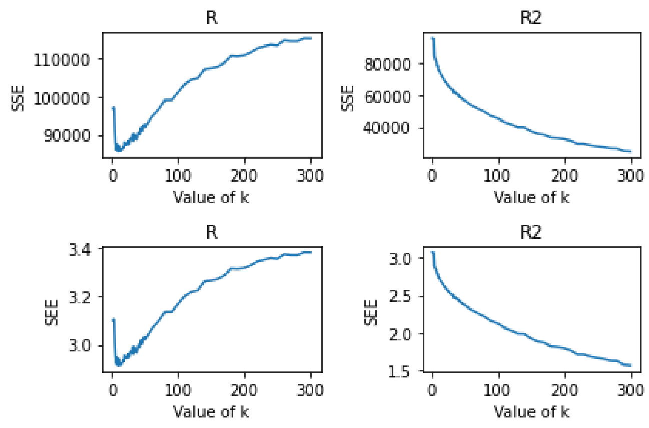

Figure 7: SSE and SEE for the NMF algorithm for the matrices R and R2

3.2 Latent Space Analysis

For each algorithm, a projection of the latent space associated with the movies was made. In figure 8 can be seen the results. Each point represents a movie, green points represent 'good movies' (with an average rating above 3.5), red point represent 'bad movies (with an average rating below 2.5) and yellow points represent the remaining ones. The blue point is the mean of the scores of the movies obtained for each algorithm.

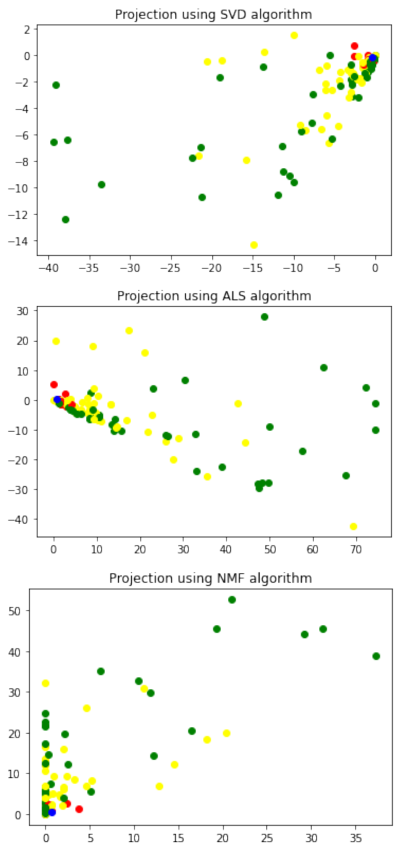

Figure 8: Scatter plot for the 2 dimensional latent space associated with the movies. Green points represent movies with an average rating above 3.5, red point represent movies with an average rating below 2.5 and yellow points represent the remaining ones. The blue point represents the mean of the scores of the movies obtained for each algorithm.

In these plots there are three visible aspects that are common for the three algorithms:

1. **One large cluster around the mean**, which can be explained by the fact that the majority of the movies have an average rating close to the mean value.

2. **Every bad rated movie is near the mean score of each projection**. One possible reason for that is that the great majority of the movies with an average classification below 2.5 has an average classification near of 2.5, as it can be seen on the histogram of figure 9.

3. **The farthest points represent generally good movies**; opposed to what happens with bad movies, there are some good movies that have an average rating that is far from the mean rating. For that reason, in the projection produced by any of the three algorithms appeared to show an outlier score.

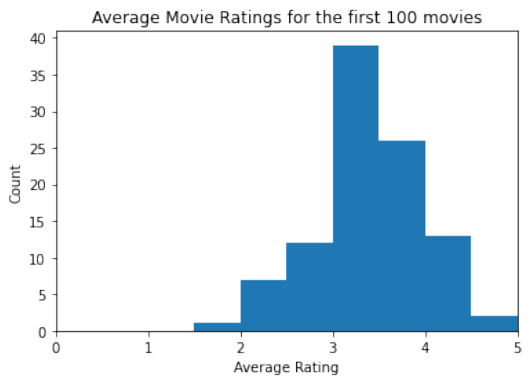

Figure 9: Histogram of the Average Rating for the first 100 movies given by the 610 users.

The latent space projection for k=2 produced by the three algorithms seems to explain some of the variability of the data, but clearly it is not good enough to give us a precise feedback about witch movies have a good score an witch ones don't. Despite of that, the projection produced by the ALS algorithm seems gives a more precise idea of the first two principal components of the data, since the points appear to distribute accordingly by two distinct directions.

4 Conclusions

The approach of using the mean value for the imputation of the missing values showed to be more precise for making predictions using the matrix factorization algorithms presented, rather the replacement of them by zeros, which in return showed to be computationally faster.

The algorithm using the Singular Value Decomposition significantly outperformed the other two, by both means of precision of the prediction and the computational cost. Even for some low rank values, the predictions computed using the mean value for the imputation of the missing values were relatively precise. This could be even used, for instance, to build a recommendation system that uses large amounts of data and requires low processing power.

More precise results could have been obtained using smaller data sets or higher processing power since for the

NMF algorithm the simulations were not done until an acceptable convergence point was reached; also, a smaller step for value of the rank would have produced more accurate knowledge about the algorithms performance.

It also would be interesting to apply the sparseness constraints to the NMF algorithm referred in [8] and see if that would translate into a relevant computational speed up of the algorithm.

The latent space analysis provided some insightful results and it made possible to visualize some relevant aspects of the data, although it's use to make predictions about the data showed to be extremely inaccurate. A higher dimension projection would probably give more useful information about the that, as the cross validation plots seem to indicate, but it would make impossible to get a visual perspective of the data.

References

[1] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[2] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[3] Gábor Takács and Domonkos Tikk. Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 83–90, 2012.

[4] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5:19:1–19:19, 2015.

[5] Dan Kalman. A singularly valuable decomposition: the svd of a matrix. *The college mathematics journal*, 27(1): 2–23, 1996.

[6] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3): 211–218, 1936.

[7] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[8] Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(9), 2004.

[9] D Seung and L Lee. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.

[10] Nathan Srebro, Jason Rennie, and Tommi Jaakkola. Maximum-margin matrix factorization. *Advances in neural information processing systems*, 17, 2004.

[11] Trevor Hastie, Rahul Mazumder, Jason D Lee, and Reza Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *The Journal of Machine Learning Research*, 16(1):3367–3402, 2015.

[12] Behnoush Abdollahi and Olfa Nasraoui. Explainable matrix factorization for collaborative filtering. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 5–6, 2016.