MovieLens Capstone Project

Dave McRae

2022-07-11

Executive Summary and Introduction

This report presents a solution to a machine learning challenge to predict 999,999 movie ratings (the test set) with a root mean squared error (RMSE) of less than 0.86490. The predictions must be made using only a train set of 9,000,055 ratings of 69,878 movies by 10,677 users. The provided test and train set are constructed from the MovieLens 10M dataset (Harper and Konstan 2015), a publicly available dataset of 10 million ratings of 10,000 movies by 72,000 users.[1] The challenge is inspired by the Netflix Prize, which ran from 2006-2009, in which the company offered a $1 million prize to the best performing team that could improve the accuracy of its Cinematch algorithm by at least 10 percent.[2] The winning team in the Netflix Prize achieved a RMSE of 0.8712 (Chen 2011).

To conceptualize this challenge, we can think of the prediction task as being to fill in the missing values in a matrix constructed from the train set, in which each user's ratings form a row, and the ratings of each movie form a column. Because the test set was constructed in such a way as to ensure that all users and movies in it are also contained in the train set, the test set ratings are a (small) subset of the more than 700 million missing values in this 10,677 x 69,878 matrix. Missing values arise when a user has not rated a movie (or, in the case of the test set, when the ratings have been excised from the dataset).

Below, we predict the test set ratings using the recosystem package for R,[3] which employs matrix factorization to produce a vector of ratings predictions for a given set of pairs of movies and users. This approach predicts the test set ratings with a RMSE of 0.77793, well below the success threshold for this challenge.

Background

This report comprises four sections. In this initial section, we introduce the dataset, its features and an example approach to the prediction task. Next, a methods section explains the matrix factorization approach and our implementation of it using the recosystem package. Third, a results section presents the outcomes of our tuning of model parameters and our ratings predictions. Finally, in conclusion, we offer some brief observations on the effectiveness of this approach and possibilities for further enhancement.

This overview of the dataset presents summary statistics for the provided train set ("edx"), comprising 9,000,055 ratings of 69,878 movies by 10,677 users drawn from the MovieLens 10M dataset. In our prediction of ratings, we make a 90:10 partition of this train set to create a new, smaller train set and a probe set for model testing. As we do not use any of these summary statistics to tune the model, however, we present them for the entire provided train set.

Table 1 shows three sample entries in the dataset. The time the review was posted (timestamp) is recorded as the number of seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970 - these three reviews were posted in 2006, 2000 and 1998 respectively. Inclusion of the year in the movie title avoids

[1] https://grouplens.org/datasets/movielens/10m/

[2] https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/

[3] https://github.com/yixuan/recosystem

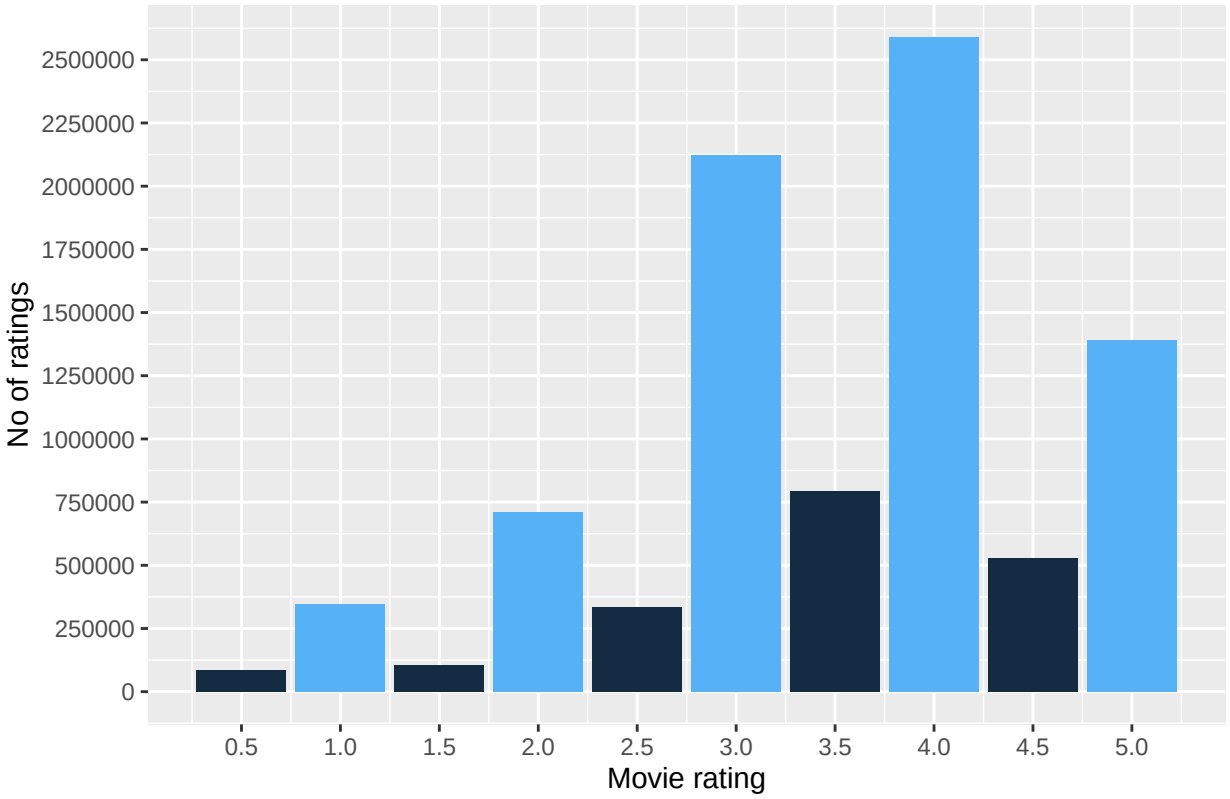
ambiguity between different movies with identical titles; we see also that multiple genres are possible for a single film.

Table 1: Sample entries, train set

userId	movieId	rating	timestamp	title	genres
124	27731	4	1138994311	NA	NA
1088	2366	5	948472675	NA	NA
10048	1608	3	888845620	NA	NA

Ratings in the dataset range from 0.5 to 5. Only full or half point ratings are possible, with half point ratings such as 1.5 less frequent than neighboring full point ratings (i.e 1 or 2 - see Figure 1).

Figure 1: No of each possible rating in the train set



Unsurprisingly, given that there are 10677 movies in our dataset, most users have not reviewed most movies. Accordingly, if we constructed a matrix in which each user's ratings form a row, and the ratings of each movie are a column, only 1.2 percent of the 746 million possible entries would contain a rating. The distribution of ratings in such a matrix is highly uneven (here we mean the presence or not of a rating, not the value of any ratings). A matrix of the 1000 most active users and the 1000 most rated movies contains ratings for 52 percent of possible entries. An equivalent matrix for the 1000 least active users and least rated movies contains ratings for just 0.06 percent of possible entries.

The values of ratings are not simply randomly distributed. Instead, we can illustrate various underlying patterns in the data. For instance - somewhat intuitively - movies that are rated more often rate better overall. By contrast, users who rate an uncommonly high number of movies tend to be somewhat more critical in their average rating. Although not plotted here, it is similarly possible to demonstrate effects on ratings for each of the categories of metadata provided with our dataset, such as movie genre, year of movie

release, and the timestamp of each rating.

Figure 2: Average rating vs times reviewed, per movie

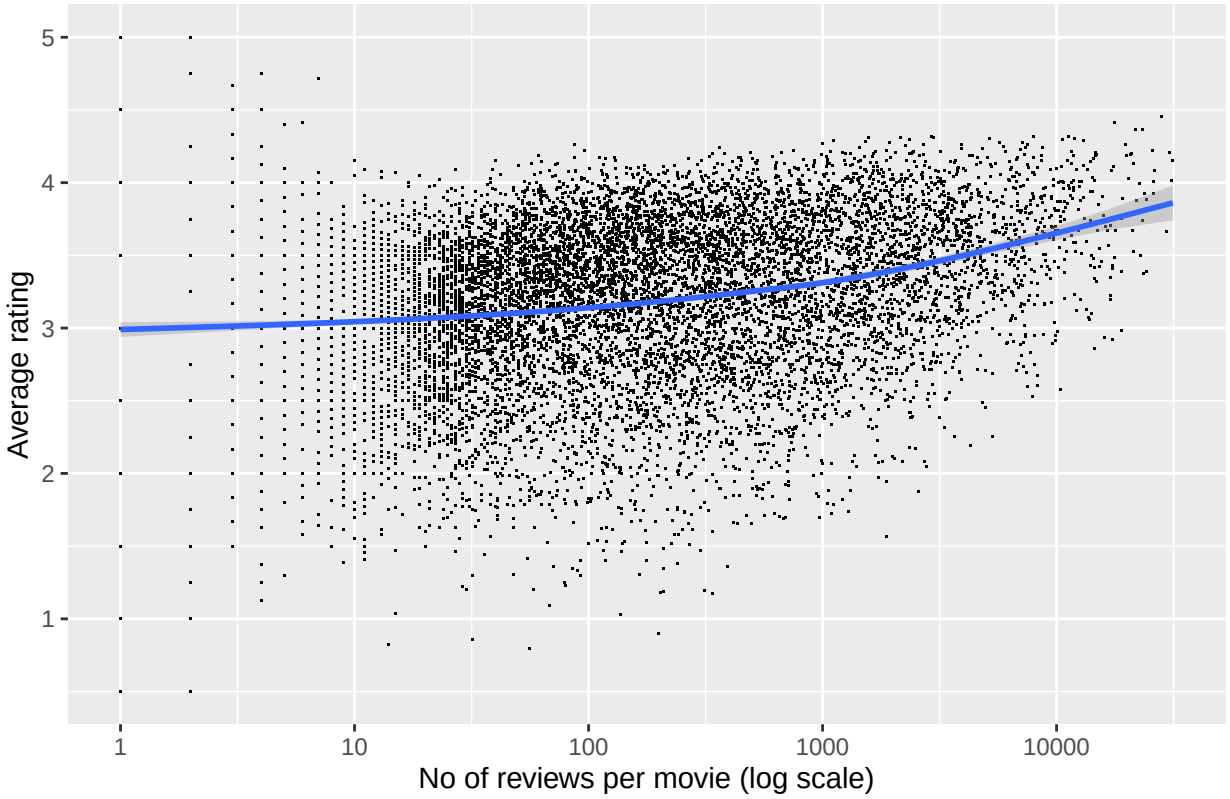
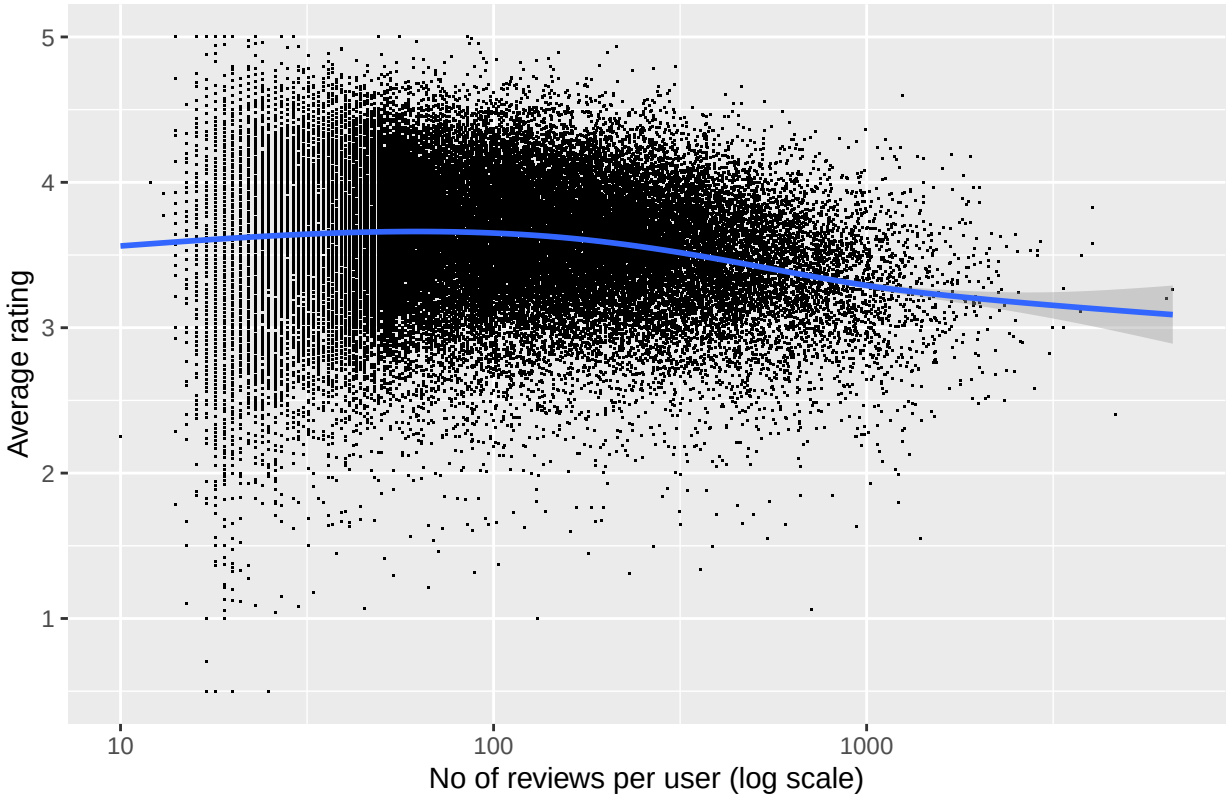


Figure 3: Average rating vs no of reviews, per user



Some ratings prediction techniques attempt to harness these effects directly to increase their accuracy. For instance, Irizarry (2022) first constructs a ratings prediction algorithm that simply predicts the dataset average rating for any missing value. If we apply this algorithm to our probe set (see Methods section) - predicting 3.5124056 for any missing value we obtain a root mean squared error (RMSE) of 1.0603, meaning the prediction for each rating on average differed from the true value by that amount.

Irizarry (2022) goes on to demonstrate that adding an item bias term for each movie to this average - where the bias term is calculated as the average rating of an individual movie minus the dataset average rating - improves the RMSE further. Additionally adding a user bias term - obtained by subtracting both the dataset mean and the item bias term from every rating by a given user, and then calculating the mean of these residuals - further decreases the RMSE.

Table 2: Performance of Irizarry's simple prediction models

Method	RMSE
Just the average	1.0603099
Movie effects model	0.9436226
Movie + user effects model	0.8663835

We could continue to decrease the RMSE by adding regularization terms to our movie and user bias, and by adding terms for more and more of the patterns that we could identify in the data. As will be discussed in the next section, however, we can also dramatically improve upon these simple prediction algorithms using matrix factorization. This technique also harnesses underlying structures in the data to improve the accuracy of predictions, but carries the advantage that we do not need to individually identify sources of bias and correlation in the data.

Methods

To generate predicted ratings, this report uses matrix factorization by employing the recosystem package for R. Matrix factorization relies on the principle that there is underlying structure in our dataset of movie ratings that creates correlations and redundancies between our rows of each user's ratings and/or our columns of the ratings of each movie (Aggarwal 2016, 92). As we canvas above, Aggarwal cites the affinity of users to particular genres and the membership of movies in these genres as two examples of underlying structure. Equally, though, these underlying structures may not be readily interpretable (Aggarwal 2016, 94, 96).

The basic principle of matrix factorization, for a m x n ratings matrix of m users and n movies, is to identify two matrices U (m x k) and V (n x k), where the k columns of U and V are each of these underlying structures. As we want to achieve dimension reduction, k is set to be less than or equal to the minimum of m and n - in practice it may be smaller by orders of magnitude than either. Each row of U contains the affinity of each user to our k structures; each row of V equivalently contains the affinities of our movies to these same structures. We can derive U and V from a sparse matrix of ratings - i.e. where most values are missing - and then construct a completed matrix of ratings predictions by multiplying U by the transpose of V.[4]

The recosystem package[5] provides a wrapper in R for LIBMF, an open source library for matrix factorization that makes use of parallel processing.[6] Recosystem is not the only recommendation system package available for R - recommenderlab is another option.[7] Recommenderlab implements a broader range of recommendation algorithms, including matrix factorization, and provides functionality to compare the performance of multiple algorithms and to created weighted hybrid models of several algorithms. For a dataset of our size on a home computer, however, the memory requirements and processing times of recommenderlab are prohibitive. Although it is not as flexible as recommenderlab, recosystem is thus preferred, as it will readily run on a home computer. Recosystem additionally provides various options to work around the memory and processor constraints of a particular system, such as saving intermediate outputs to disk instead of memory.

The data analysis procedure using recosystem was as follows:

1. Under the challenge, a train set ("edx") and a test set ("validation") were provided as dataframes. The test set was constructed as a random ten percent sample of the train set, with any users or movies that occurred only in the test set returned to the train set. Each observation in the dataframes consists of a user ID, a movie ID, the title of the movie (including its year of release), the rating by the user, a timestamp for the rating, and a list of the genres for the movie. In the following analysis, the test set is set aside for final validation only, with all model tuning and fitting performed only on the train set.

2. As a first step, the provided train set ("edx") was partitioned again by the identical procedure used to create the test set ("validation"), to create a new train set ("edx_train") and a probe set ("edx_probe"). The probe set is temporarily set aside during model tuning and initial training to test the accuracy of the model, as measured by root mean squared error (RMSE).

3. We use recosystem's $tune function to tune a matrix factorization model on edx_train. The function employs five-fold cross-validation to tune the number of latent factors, regularization costs for user factors and item factors, and the learning rate. For the model used in this report we use a single thread version of the code to ensure replicability. [8] Value ranges used for tuning and the tuned parameters are reported in the results section below, we specified ten iterations for model tuning.

[4] The explanation of matrix factorization in this paragraph is adapted from Aggarwal (2016, 94)

[5] https://cran.r-project.org/web/packages/recosystem/index.html

[6] On the LIBMF library, see Chin et al (2016)

[7] https://github.com/mhahsler/recommenderlab

[8] Using four parallel processing threads to suit our computer's multicore processor greatly lessened processing time and converged to the same tuned parameters, but produced a different error from the loss function each time. This may result from recosystem or the underlying LIBMF library using random number generation during parallelization not controlled by R's "set.seed" function. See https://stat.ethz.ch/pipermail/r-help/2021-August/471875.html

4. Using the optimized parameters, we then trained the model on edx_train using recosystem's $train function. This function uses an iterative process of gradient descent to produce matrices P and Q, where P multiplied by the transpose of Q predicts each of our missing ratings.

5. We test our trained model on the probe set ("edx_probe") to obtain an indicative RMSE. As our probe set was not used at all in development of the model, and was constructed in the same way as our test set, there is no reason to expect the model to perform worse on our test set, beyond variations between random samples. If the indicative RMSE is below the success threshold for the machine learning challenge - with some margin of error for sampling variations - we proceed to train the model on our full train set ("edx"), using the parameters as optimized on edx_train. With this final retraining step our model may actually perform better on the test set than on our probe set, as it has now been trained on a larger dataset.

6. We use the model as trained on our entire train set to predict the test set ratings.

Results

Model Tuning

The table below shows the ranges of values used in tuning our model using recosystem's $tune function, as well as the final tuned parameters. By default, $tune's loss function outputs a squared error - it was 0.79627 for the best performing set of parameters and 1.06039 for the poorest performing set of parameters.

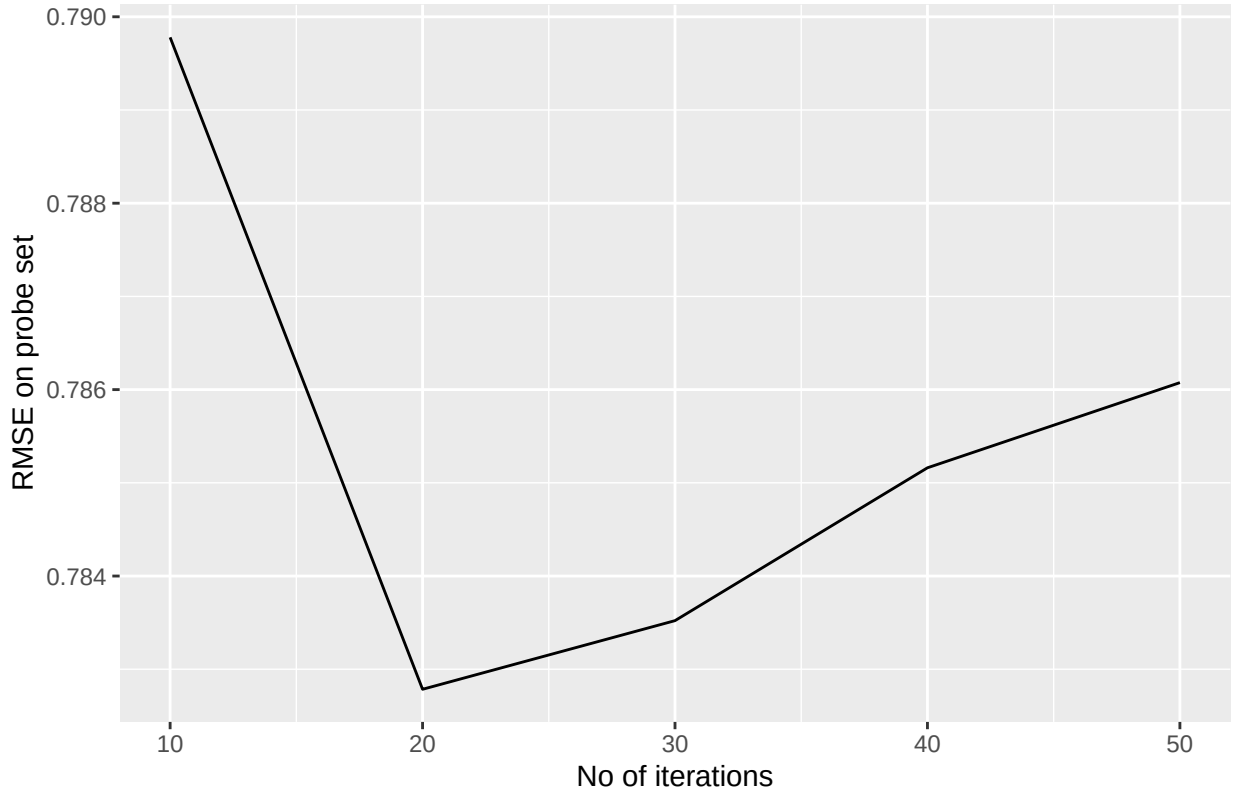
Table 3: Parameters used for tuning and final tuned settings

Parameter	Description	Status	Range	Tuned Parameter
dim	Number of latent factors	User-defined	10, 20, 30, 40, 50	50
costp_l1	L1 regularization cost for user factors	Default	0, 0.1	0
costp_l2	L2 regularization cost for user factors	Default	0.01, 0.1	0.01
costq_l1	L1 regularization cost for item factors	Default	0, 0.1	0
costq_l2	L2 regularization cost for item factors	Default	0.01, 0.1	0.1
lrate	Learning rate/step size in gradient descent	User-defined	0.1, 0.2	0.1

Training the model and testing on the probe set

Using the tuned parameters, we trial 10, 20, 30, 40 and 50 iterations of gradient descent (see figure below). We find 20 iterations produces the lowest RMSE on the probe set. As the probe set RMSE is well below the success threshold for this challenge, we proceed to retrain our tuned model on the entire train set, using 20 iterations, and predict our test set ratings.

Figure 4: Probe set RMSE per training iterations



Final Training and Predictions

Using the model to predict the test set ratings, we obtain a RMSE of 0.77793.

Conclusion

Matrix factorization, as implemented using the recosystem package in R, provides an accurate and efficient method to predict ratings even when only a small proportion of user ratings are known. If further improvements in prediction accuracy were required, one avenue to explore may be to ensemble other prediction methods alongside matrix factorization, as Irizarry (2022) explains that combining different prediction algorithms generally improves results. An ensemble approach would also be consistent with the winning entry in the Netflix prize, which ensembled more than 500 methods to produce its final prediction (Chen 2011). In practice, though, matrix factorization was more than sufficient to achieve the success threshold for this machine learning challenge.

References

Aggarwal, Charu C. 2016. *Recommender Systems: The Textbook*. Switzerland: Springer Cham. https://doi.org/10.1007/978-3-319-29659-3

Chen, Edwin. 2011. 'Winning the Netflix Prize: A Summary'. http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

Chin, Wei-Sheng and Bo-Wen Yuan and Meng-Yuan Yang and Yong Zhuang and Yu-Chin Juan and Chih-Jen Lin. 2016. 'LIBMF: A Library for Parallel Matrix Factorization in Shared-memory Systems.' *Journal of*

Machine Learning Research 17, no 86, 1-5. https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_open_source.pdf

Harper, F. Maxwell and Joseph A. Konstan. 2015. 'The MovieLens Datasets: History and Context.' *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5, no 4, Article 19. http://dx.doi.org/10.1145/2827872

Irizarry, Rafael A. 2022. *Introduction to Data Science: Data Analysis and Prediction Algorithms with R* https://rafalab.github.io/dsbook/