PCML Project 2 - Recommender System for Netflix Prize Dataset

Manana Lortkipanidze, Aidasadat Mousavifar, Frederike Dümbgen
EPFL, Switzerland

***Abstract*—In this course project, a recommender system for the Netflix Prize dataset consisting of ratings of 1'000 users for 10'000 movies is implemented and tested. First, latent feature models are proposed using stochastic gradient descent or alternating least squares for the matrix factorization. Secondly, neighborhood and baseline models such as k-nearest neighbors are added to capture similarities between users and/or items for better predictions. ALS scores best in terms of rmse (0.98418) and promising results for linear blending of different methods are presented (0.98490).**

I. Introduction

In the age of online platforms for entertainment, shopping and knowledge transfer, the automatic characterization of the available items and the involved users has become a crucial factor of success. The tools created for such purposes are commonly called "recommender systems". In some cases, the recommendations can be done using additionally provided information on users and items, such as demographic measures like age and gender or official popularity measures and item classifications. This content-based approach tends to be too simplistic and cumbersome for large databases and adds a dependeny on third-party data of unknown accuracy. Methods grouped under the term "collaborative filtering" overcome these difficulties by predicting behaviour based on past feedback. Feedback can either be of implicit form e.g. how many times did a user click on an item, how long did he watch a TV series, etc., or of explicit form, such as rankings or feedback form surveys [1]. When users and items are grouped by their similarity and recommendations are done based on similar users (user-oriented models) or similar items (item-oriented models) respectively, one talks about neighborhood models. If, on the other hand, the items and users are characterized by a set of K features whose signification is a priori unknown, one talks about latent factor models.

The goal of this project is to apply collaborative filtering to the Netflix dataset, which consists of explicit feedback in terms of ratings of $D = 1000$ users and $N = 10000$ items. The main focus is on latent factor models, but neighborhood models can be taken into account to improve predictions.

II. Models and Methods

A. Preprocessing

1) Data analysis: The data provided is of the form

$$\mathbf{X} = (x_{nd}), \tag{1}$$

where x_{nd} corresponds to the rating of user d for movie n. the ratings are discrete and lie between 1 and 5. Since every user only rates a very small subset of movies, the matrix is sparse. The data have the following characteristics:

- The trainig set provided has 991561 non-zero entries, denoted by $R(u, i)$ which corresponds to a fraction of ≈ 0.1 with respect to the number of unknowns ($N \times$D).
- Each user in the dataset has rated at least 8 movies, the maximum ratings per user is 4590. On average, a user rates roughly 1177 movies.
- Each movie in the dataset is rated by at least 3 users, and the maximum of ratings per movie is 522. On average, each move is rated by around 118 users.

2) Splitting: The provided training data has been split into a training set and 2 small validation sets, as described more in detail in II-D. k-fold cross-validation for was performed on the training set.

3) Bias correction: Users and movies can vary a lot in terms of their average ratings, consequently, there might be some implicit bias in the data provided. This biais was removed as follows by subtracting a correcting term from each element,

$$\widetilde{x}_{nd} = x_{nd} - \mu_{nd} \tag{2}$$

$$\mu_{nd} = \begin{cases} \mu_n = \frac{1}{|R(n)|} \sum_{d \in R(n)} x_{nd}, & \text{for item bias only} \\ \mu_d = \frac{1}{|R(d)|} \sum_{n \in R(d)} x_{nd}, & \text{for user bias only} \\ \mu = \frac{1}{|R(n,d)|} \sum_{n,d \in R(n,d)} x_{nd}, & \text{for global bias only} \\ \mu_n + \mu_d - \mu & \text{for combined biais} \end{cases}, \tag{3}$$

where the combined bias term corresponds to what is used in [2]. The entries of the residual matrix, $\widetilde{x}_{nd}$ are then factorized as explained in II-B and the bias is added again for the final predictions (Figure 1).

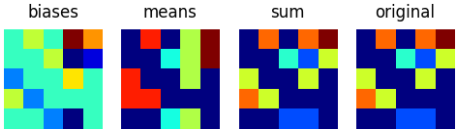


Figure 1: A simple example for bias correction. The bias matrix is obtained by substracting the mean of the corresponding user. The matrix composed of these means and the biases sum up to the original training matrix as expected.

The performance with and without corrections was tested

on ALS (Figure 2. The combined bias with underlying assumption that each rating can be composed of a contribution by the user and one by the item, is the most accurate model.

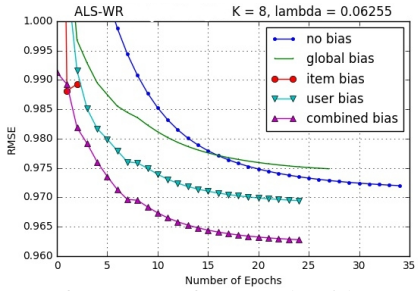


Figure 2: Performance of ALS algorithm using different variants of bias correction as shown in (3)

B. Latent Feature Models

In order to fill the missing ratings, we can factorize the ratings matrix by two low-rank matrices,

$$\mathbf{X} = \mathbf{W}\mathbf{Z}^T, \qquad \text{with} \quad \mathbf{W} \in \mathbb{R}^{N\times K}, \mathbf{Z} \in \mathbb{R}^{D\times K}, \tag{4}$$

where K is the number of latent features, and $\mathbf{Z}$ and $\mathbf{W}$ are in the following referred to as the user and feature matrix respectively.

1) Bias Stochastic Gradient Descent: The stochastic gradient descent (SGD) algorithm used in this report is provided by the *Surprise* library [3]. It is based on the algorithm provided by Simon Funk [1], which uses bias correction as described in [2], which minimizes the cost function as defined in (9). Without this bias correction, it reduces to the algorithm given in [4].

As shown in II-A3 and by [5], we expect better predictions when removing the user and item biases, so we choose to use this correction and we choose the other hyperparameters using cross-validation.

2) Alternating Least Squares: Alternating Least Squares (ALS) algorithm used in this report is implemented by us. Each step of this method is roughly following: consider W as a constant and find optimal Z by using normal equations and vice versa.

To prevent our model from overfitting we used regularized ALS, more specifically Tikhonovs relugarizer .

$$\mathcal{L}(W,Z) = \sum_{(i,j)} (r_{ij} - W_i Z_j^T)^2 \tag{5}$$

$$+ \lambda(\sum_i n_{W_i}||W_i||^2 + \sum_j n_{Z_j}||Z_j||^2) \tag{6}$$

,where W_i are ratings receveid by the i-th movie and Z_j are ratings given by the j-th user, n_{W_i} and n_{Z_i} are number of ratings received by the i-th movie and number of movies rated by the ith user respectively, λ is regularizations parameter and r_{ij} is rating recevied by the i-th movie from the j-th user.

Reason behind using such regularizations is, that [6] claims it prevents model from overfitting. Our observation is, that it works well when number of laten features is below 50, otherwise there is almost no improvement.

We observed that solving normal equations for single rows of $\mathbf{W}$ or $\mathbf{Z}$ do not depend on each other, which gives us possibility to seperate them.As mentioned above, data matrix is incomplete so in order to solve normal equations for rows of $\mathbf{W}$ or $\mathbf{Z}$ we create small subset of data, consisting only of ratings we need and corresponding rows of $\mathbf{W}$ or $\mathbf{Z}$, depending for which matrix we solve the equation.

$$(W_i W_i^T + \lambda n_{Z_i}) Z_i = W_i R_{ic\_nnz} \tag{7}$$

$$(Z_i Z_i^T + \lambda n_{W_i}) W_i = Z = W_i R_{ir\_nnz}^T \tag{8}$$

,where R_{ic_nnz} and R_{ir_nnz} are ratings(nonzero) given by the i-th user and ratings(nonzero) received by the i-th movie respectively.

C. Neighborhood Modles

1) Baseline methods: Large user and item effects—systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. It is customary to adjust the data by accounting for these effects, which we encapsulate within the baseline estimates. Denote by μ the overall average rating. A baseline estimate for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects: $b_{ui} = \mu + b_u + b_i$. The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. In order to estimate b_u and b_i one can solve the least squares problem:

$$min_b \sum_{(u,i)\in K} (r_{ui} - \mu - b_u - b_i)^2 + \lambda(\sum_u b_u^2 + \sum_i b_i^2). \tag{9}$$

Here, the first term strives to find b_u and b_i that fit the given ratings. The regularizing term avoids overfitting by penalizing the magnitudes of the parameters. Baselines can be estimated in two different ways: using Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS), both are implemented by [3].

2) kNN: k-nearest neighbors (kNN) are a classical method for recommender systems. Ratings or items are predicted by using the past ratings of the k most similar users or items. If the influence of the similar users/items is weighted by the similarity, all users/items may be used for the prediction. Using the user-item matrix to compute the similarity is often called collaborative filtering. Computing the item similarities from the item attributes leads to content-based filtering. Popular similarity metrics are the cosine

[1] http://sifter.org/ simon/journal/20061211.html

similarity, Pearson correlation, Mean Squared Difference similarity, and baselines based pearson correlation.

We use flowing methods to predict the ratings, implemented in [3]. All formulas are based on user similarities and it is same for as item similarities based formulas.

- KNNBasic

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u,v).r_{vi}}{\sum_{v \in N_i^k(u)} sim(u,v)}$$

- KNNWithMeans

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} sim(u,v)(r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} sim(u,v)}$$

- KNNBaseline

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_i^k(u)} sim(u,v)(r_{vi} - b_{vi})}{\sum_{v \in N_i^k(u)} sim(u,v)}$$

They are For each of these algorithms, the actual number of neighbors that are aggregated to compute an estimation is necessarily less than or equal to k. First, there might just not exist enough neighbors and second, the sets $N_i^k(u) N_k^u(i)$ only include neighbors for which the similarity measure is positive. It would make no sense to aggregate ratings from users or items that are negatively correlated.

D. Blending

It has been shown that if one disposes of several well performing methods, a clever combination of the methods can improve the accuracy of predictions. A combination can overcome problems when the used algorithms are prone to converge to local optima, or full convergence is too computationally expensive. [7] A combination of multiple algorithms can be particularly beneficial when some are shown to perform better in certain regions of the features space than others, giving each method a higher weight in the regions where it performs best.

The splitting of the kaggle dataset is done as suggested in [8]: each method is trained on a fixed subset of 95% of the dataset (training set), while the remaining 5% are split up equally, and one half is used to compare the different methods (probe set) and the other half is used to create a valuable prediction of the performance of the final, blended method (test set). This splitting ensures at least 30'000 non-zero entries in the smallest probe and test sets, which was considered enough for the evaluation, and removes only a small proportion of the valuable training data.

The process of linear blending is shown in Figure 3. The blending model is obtained by minimizing the mean squared prediction error on the probe set as suggested in [8]. If we call p_i the vector of N_P ordered predictions of method i ($i = 1 \dots N_M$) on the probe set, and r the true values (in the same order), then the weight vector x is obtained by solving

method	test	kaggle
SGD	1.0002	0.99256
ALS	1.02654	0.98518
SGD Baseline	1.00059	-
ALS Baseline	0.99877	-
KNN ALS item	1.04101	-
KNN ALS user	1.04285	-
Blending of above	0.98655	0.98490

Table I: Comparison of results obtained with different algorithms.

$$\mathbf{P} = [p_0, p_1, \dots, p_{N_M}] \in \mathbb{R}^{N_P \times N_M}, \tag{10}$$

$$x = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T r, \qquad \text{with} \quad x, r \in \mathbb{R}^{N_M}. \tag{11}$$

The best prediction is obtained by applying these weights to the predictions of the methods on the kaggle set of size N_T, q_i, yielding the optimum ratings vector,

$$\mathbf{Q} = [q_0, q_1, \dots, q_{N_M}] \in \mathbb{R}^{N_T \times N_M}, \tag{12}$$

$$\hat{q}_i = \mathbf{Q}x \in \mathbb{R}^{N_T}. \tag{13}$$

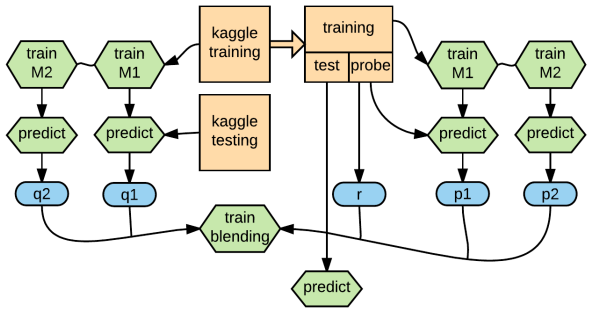


Figure 3: Visualization of blending process used to combine the methods (**orange**: datasets, **green**: operations, **blue**: results). The models are trained on a subset of the kaggle training set and predictions tested on the probe set to evaluate their performance (p_i vs. r). The predictions (q_i) for the kaggle test set are optimally weighted using linear regression, and the obtained blending method is tested on the test to predict its performance.

III. Results

The results for all tested methods are shown in Table I.

The parameters for SGD are found using 3-fold cross-validation. The best set of parameters in terms of test error found is $\lambda = 0.0359$, $\gamma = 0.005$, $K = 100$, $N_{epochs} = 28$, as shown in Figure 4.

A visualization of the obtained predictions was created by assembling all non-zero entries for each user and coloring each element based on the ratings (Figure 5).

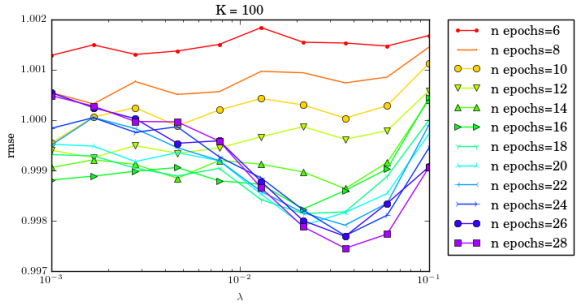
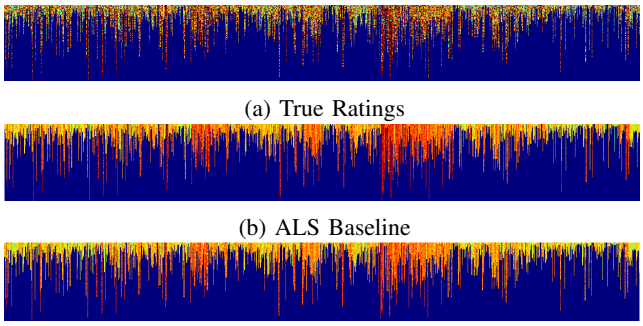


Figure 4: Results from 3-fold cross validation for different parameters of SGD matrix factorization.



(a) True Ratings

(b) ALS Baseline

(c) Linear Blending

Figure 5: Assembled ratings of different methods, colored by their respective values.

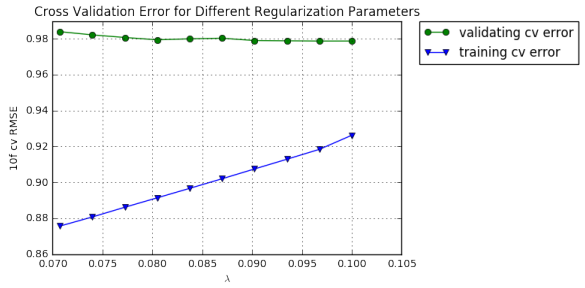


Figure 6: Results of 10-fold cross-validation (average rmse) for ALS

ALS with weighted regularization gave us better results, compared to kNN and baseline methods. Models with more than 100 latent features were overfitting easily but giving good results in case of early stoppig (500 features scored around 0.988 on kaggle). The best results were achieved with $K = 8$ (Figure6)

visualizes results of 10-fold cross-validation, for different λ-s. 25 latent features scored slightly more than 8 features, Figure7 plots change of RMSE train and RMSE test with respect to the step size, for different λ-s. Experiments with latent features above 500 proved inneficient, mostly because of overfitting.

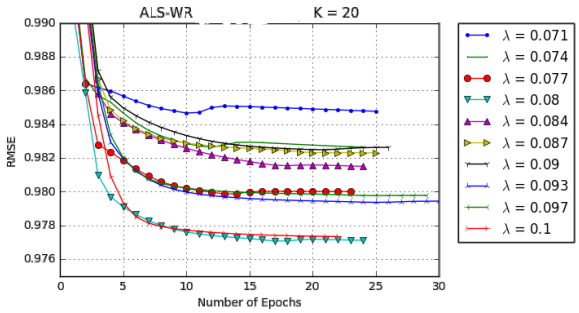


Figure 7: Run of different λ for test=10%, train=90%.

	startup	ease of use	performance	variety
graphlab [9]	★★	★★★	★	★★
pyspark [10]	★	★	★★	★★★
surprise [3]	★★★	★★★	★★★	★★
fancyinput [11]	★★★	★★★	★★	★

Table II: Overview of libraries tested for recommender system implementation. One star corresponds to lowest performance.

A. *Technical Details*

1) Library Review: Different libraries have been tested for the implementation of the methods, an overview is shown in Table II. The surprise library was chosen for its high performance thanks to customizability, great ease of startup (installation & system requriements), ease of use and big variety of implemented algorithms for recommender systems.

IV. Discussion

Both ALS and B-SGD show a good performance on the dataset in terms of rmse error. Because of the high number of hyparameters, especially the need for an appropriate step size, B-SGD is the more difficult method to tune. ALS is easier to implement, however each step is more expensive, which makes it slower. To sum up with, both methods can achieve equally good results as long as the parameters are tuned optimally.

When looking more closely at the predictions, one can see that the prediction vary very little from the average prediction of around 3. This can be explained by the choice of the squared loss function: a wrong prediction is very strongly penalized by this loss function, so in case of uncertainty the method tends to predict the ”safer” average values. This effect can be compensated for by adding similarity-based methods such as k-nearest neighbours. Blending these methods with the matrix factorization methods in an optimal way leads to more varied and accurate predictions.

V. Conclusion

Both latent factor models and neighborhood models have been trained for the collaborative filtering problem proposed in the Netflix Grand Prize competition. The matrix

factorization for the latent factor models was done using ALS and B-SGD, with the number of latent features and the optimization parameters determined empirically using k-fold crossvalidation. The best prediction was obtained using ALS, but B-SGD improved similarly well. The blending of these methods with neighborhood models such as item- or user-based KNN has proven promising, leading to more varied predictions.

REFERENCES

[1] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets Yifan," *IEEE International Conference on Data Mining*, pp. 263–272, 2008.

[2] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," *Computer*, vol. 42, no. 8, pp. 42–49, 2009.

[3] Nicolas Hug, "Surprise." [Online]. Available: https://github.com/NicolasHug/Surprise/releases

[4] R. Salakhutdinov and A. Mnih, "Probabilistic Matrix Factorization," in *Neural Information Processing Systems 21*, vol. 25, no. 2005, 2008, pp. 880–887.

[5] C. R. Aberger, Y. Koren, R. Bell, and C. Volinsky, "Recommender : An Analysis of Collaborative Filtering Techniques," *Computer*, vol. 42, no. 8, pp. 42–49, 2009.

[6] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, *Large-Scale Parallel Collaborative Filtering for the Netflix Prize*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–348. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68880-8{_}32

[7] T. G. Dietterich, "Ensemble Methods in Machine Learning," *Multiple Classifier Systems*, vol. 1857, pp. 1–15, 2000. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-45014-9{_}1

[8] T. Andreas, M. Jahrer, R. M. Bell, and F. Park, "The BigChaos Solution to the Netflix Grand Prize," pp. 1–52, 2009.

[9] Carnegie Mellon University, "Graphlab collaborative filtering." [Online]. Available: http://www.select.cs.cmu.edu/code/graphlab/pmf.html

[10] Apache Spark, "Spark python api." [Online]. Available: http://www.select.cs.cmu.edu/code/graphlab/pmf.html

[11] Alex Rubinsteyn, Sergey Feldman, "fancyimpute." [Online]. Available: https://pypi.python.org/pypi/fancyimpute/