EPFL ML Recommender System - Project 2

Alberto Chiappa, Tasho Kjosev and Riccardo Peli (Group Name: $\#noChill$)

Abstract— **In this paper we propose a model for a recommender system to predict evaluations by users for items, in this case movies.**
We have structured this paper in different sections. After an introduction in section I, in II we describe the dataset and the aim of this paper. In III we present in details the algorithms that we used to build the model and how we have validated them. In IV we discuss the results and we comment about the goodness of the predictions. In particular, we show that combine different approaches is a good strategy to obtain a accurate model.

I. INTRODUCTION

The aim of this project is to predict user ratings for movies, based on already known ratings, without any additional information. This kind of systems is useful in many applications, in which users grade items, like movies, books or restaurants. In this paper we present several methods to acquire recommender systems and we compare their performances in terms of accuracy. To do that, we use the Root Mean Squared Error (RMSE from now on) defined as

$$RMSE := \sqrt{\frac{1}{|\Omega|}\sum_{(d,n)\in\Omega}(x_{dn}-\widetilde{x}_{dn})^2} \qquad (1)$$

where Ω is the set of nonzero data, d is the index of the movies, n is the index of users, x is the data and $\widetilde{x}$ is the approximation that is provided by the model.

II. DATASET

The dataset is composed by a list of indices $(user, item)$ and the evaluation of that movie from that user, an integer number between 1 and 5. There are $D = 10000$ movies and $N = 1000$ users, but just 1176952 evaluations are available (approximately 11.7%). The aim of the project is to find a model that can accurately predict the missing evaluations.

III. MODELS AND ALGORITHMS

In this section we describe the methods and the algorithms we used to produce our model. We explain how we validated and we tuned the parameters for each of them.

A. Baselines

Before building any of the models, we needed a baseline to which to compare our other methods.

1) Global Average: We predicted all the values with the global average of the data. The score on Kaggle is 1.11820.

2) Movie Average: Given the postion (d, n) we predicted the value with the average of the ratings of the movie d. The score on Kaggle is 1.02982

3) User Average: Given the postion (d, n) we predicted the value with the average of the ratings of the user n. The score on Kaggle is 1.09268

4) Smart Average[2]: We computed the average rating for each movie and added the average offset for each user. That is:

$$\widetilde{x}_{dn} = Mean_d + Offset_n$$

However, taking that this average is just the average of the scores for a movie can lead to problems when the movie has few ratings. What we like to do is to find the true distribution from which these ratings are generated and get *its* average. This distribution should be a linear blend between the distribution of the ratings of the movie and the distribution of the averages of all movies with a blending ratio equal to the ratio of variances. That is:

$$Mean = \frac{GlobalAverage * K + \sum ObserverdRatings}{K + N_{ObverevedRatings}}$$

where

$$K = \frac{VarianceOfMovieRatings}{VarianceOfAverages}$$

The offset values are generated in the same manner.

B. Collaborative filtering[3]

Collaborative filtering, explained simply, is a technique that predicts the rating for an item by looking at items similar to it. More precisely, to predict what user n would rate the movie d we would have the following:

$$\widetilde{x}_{dn} = Avg_n + \frac{\sum_{u'\in U} sim(n,n') * (\widetilde{x}_{dn'} - Avg_{n'})}{\sum_{u'\in U} sim(n,n')}$$

Where Avg_n is the average rating of user n, and U is the set of users n' s.t. $sim(n, n') > 0$.
For calculating the similarity we use Pearson correlation, defined as:

$$sim(n,m) = \frac{\sum_{i\in I_{nm}}(\widetilde{x}_{ni}-Av_n)(\widetilde{x}_{mi}-Av_m)}{\sum_{i\in I_{nm}}(\widetilde{x}_{ni}-Av_n)^2\sum_{i\in I_{nm}}(\widetilde{x}_{mi}-Av_m)^2}$$

This way we get the user-based prediction for this user-movie pair. We can also get the item-based prediction by inverting the users and items and performing the same steps.

C. Factorization methods[1]

Given $X \in \mathbb{R}^{D\times N}$, which is the matrix of the data, we want to factorize it, i.e. find two matrices $W \in \mathbb{R}^{K\times D}$ and $Z \in \mathbb{R}^{K\times N}$ such that $X \approx W^TZ$. K is the number of features. The aim is to find such W and Z that minimize the RMSE defined in (1), with $\widetilde{x} = W^TZ$. An important aspect is that the cost function is not convex, so it is not guaranteed that the global minimum is unique, but it is convex with respect to W and Z separately. To perform the minimization we used two different optimization algorithms: Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS) .

1) Stochastic Gradient Descent: As for many optimization problems, gradient descent is an option to try to reach the minimum of the cost function. However, classic gradient descent has one main downside when applied to our problem: the matrix has 10 million elements, which makes it computationally expensive to perform any global operation on the matrix. For this reason, it is more convenient to apply a stochastic version of the gradient descent, which has the very important property of being local.
As illustrated in the lessons, the cost function (1) can be differentiated with respect to each element of the two matrices W and Z in the following way:

$$\frac{\partial}{\partial w_{k,d'}} f_{d,n}(W,Z) = -\left[x_{d,n} - \left(W^T Z\right)_{d,n}\right] z_{k,n}\delta_{d,d'}$$

$$\frac{\partial}{\partial z_{k,n'}} f_{d,n}(W,Z) = -\left[x_{d,n} - \left(W^T Z\right)_{d,n}\right] w_{k,n}\delta_{n,n'}$$

where

$$f_{d,n} = \frac{1}{2}\left[x_{d,n} - \left(W^T Z\right)_{d,n}\right]^2$$

is the (d,n) term of the sum over all the valid ratings which the cost function consists of.
The fact that the coefficient $x_{d,n} - \left(W^T Z\right)_{d,n}$ is independent of k can be exploited to update a whole column with one operation, using Numpy library's tools for matrix operations. In addition to this, the coefficient being the same for both the derivative with respect to $\partial w_{k,d'}$ and $\partial z_{k,n'}$ allows to save some computation. Despite these good points, one drawback of the method is the fact that Python is slow at performing 'for' cycles, and our implementation of this algorithm iterates through all the non-zero elements of X.
The algorithm, as it is, did not perform very well, most likely because the cost function is not globally convex. Therefore, a regularization term was required.
We simply chose to introduce a local regularization to each update of a column of W and Z, by adding to the gradient a multiple of the column itself, weighted with a parameter lambda (constant for each matrix). Although from an algorithmic point of view it has a clear meaning, the interpretation of this regularization, when isolated in the cost function, is not so immediate. The new form of the function can be computed with little calculus, and results as follows:

$$\sum_{(d,n)\in\Omega}\left(\left[x_{dn} - \left(W^T Z\right)_{dn}\right]^2 + \frac{\lambda_{it}}{2}\|W_d\|_2^2 + \frac{\lambda_{us}}{2}\|Z_n\|_2^2\right)$$

where W_d is the d^{th} column of W and Z_n is the n^{th} column of Z, while λ_{it} and λ_{us} are the two regularization parameters.
The algorithm was quite promising from the beginning, performing better with a quite large (about 25) number of features . It is very sensitive to the choice of the parameters, which required a good amount of cross-validation to perform at its best. To help convergence, the step length is reduced by 1.2 at the end of each iteration, starting from an initial step size of 1.2. We initialized the two matrices with small random numbers except from the first column, equal to the mean by movies for W and to the mean by users for Z.

2) Alternating Least Squares: An alternative approach to find W and Z is using normal equations.
Equating to zero the gradient of $RMSE$ with respect to Z with W fixed, we end up with:

$$Z = \left(WW^T + \lambda_{us} I_k\right)^{-1} WX \tag{2}$$

Conversely, equating to zero the gradient of $RMSE^2$ with to respect to W with Z fixed, we end up with:

$$W = \left(ZZ^T + \lambda_{it} I_k\right)^{-1} ZX^T \tag{3}$$

where λ_{us} and λ_{it} are parameters that come from a regularizer term $\lambda_{it}\|W\|_{Frob}^2 + \lambda_{us}\|Z\|_{Frob}^2$ added to the cost function . This term is fundamental to ensure that the matrix to invert is not singular and to have a more regular function to optimize. Now we have all the ingredients to describe the optimization algorithm:

- initialize W and Z
- iterate until convergence
  - update Z according to (2)
  - update W according to (3)
- compute X as $W^T Z$

We initialized the Z and W in the same way as in SGD. We use as stop criterion a difference in RMSE between previous and next iteration less than 10^{-5}. This algorithm is efficient, because using the Numpy library, it is easily parallelized. This results in a faster computation respect to SGD: running 30 iterations of SGD takes roughly 10 times more than 30 iterations of ALS.

D. Ensemble

Being inspired by the "Netflix Prize" competition, where combination of different methods proved to perform more effectively than the best individual methods (if properly weighted), we decided to see if this were true also in our case. We combined the two different ways to compute the matrix factorization, assigning heavier weight to the results that were better performing, and at the first try our score improved. Then, as the "ensemble" method looked promising, we decided to train the weights for the different methods in a more systematic way. First of all, we split the data matrix in 5 randomized different ways, with a 1/9 ratio between test and train. We then computed the predictions for the valid ratings stored in the test matrices with all our individual methods. At this point, we merged the 5 predictions on the tests and we were left with 8 estimates, one per method. We built a matrix with 8 columns and a number of rows equal to the number of elements in each prediction, called $|\Omega'|$, and a vector with a number of elements equal to the number of rows of the matrix. Then we filled the matrix with the predictions and the vector with the real values. The resulting system can be represented as follows:

$$\begin{bmatrix} SGD_1 & ALS_1 & AVG_1 & \dots & CFU_1 & CFI_1 \\ SGD_2 & ALS_2 & AVG_2 & \dots & CFU_2 & CFI_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ SGD_{|\Omega'|} & ALS_{|\Omega'|} & AVG_{|\Omega'|} & \dots & CFU_{|\Omega'|} & CFI_{|\Omega'|} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_8 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{|\Omega'|} \end{bmatrix}$$

where the predictions are collected in the left matrix, the exact values on the right and the weights are the unknowns. To solve the resulting overdetermined system we simply used the normal equations for the least square algorithm, being the system not too large for such an approach. The solution is the vector of the weights with which we combined the results of the different training methods.

IV. Results

In this section we present our results and how we validate the models.

A. SGD

Stochastic Gradient Descent turned out to be a very sensitive method with respect to parameter choice. Preliminary investigation revealed that the best results were achieved with a larger number of features than with Alternating Least Square. After some tests, we fixed the number of features to 25. We then iteratively cross-validated λ_{user} and λ_{item} in order to get them as close as the optimum as possible. The first explorative iterations are shown in Fig. 1 and Fig. 2.
Splitting the data assigning 90% of the features to the train data and 10% to the test data lead to the choice of $\lambda_{user} = 0.02$ and $\lambda_{item} = 0.24$
With these parameters, the prediction achieved a RMSE equal to 0.97894 on Kaggle.

B. ALS

For this method the parameters that we had to select were λ_{us}, λ_{it} and K (the number of features).
To select them, while testing for one parameter, we kept the others fixed. For each of these combinations we run a cross validation on one of them, splitting the data in train, with 90% of the data, and test, with 10%. So we learned the model using the train and we computed the RMSE on the test. In this iterative process we tried to get closer to the minimum RMSE on the test.
The results are shown in figures 3 and 4. We picked the best lambdas, as close as possible to the minimum of the RMSE on the test set. As expected, the train error increases as a function of lambda, whereas the test error exhibits a minimum. We find as optimal values:

- $\lambda_{us} = 31.8553$
- $\lambda_{it} = 20.05672522$
- $K = 10$

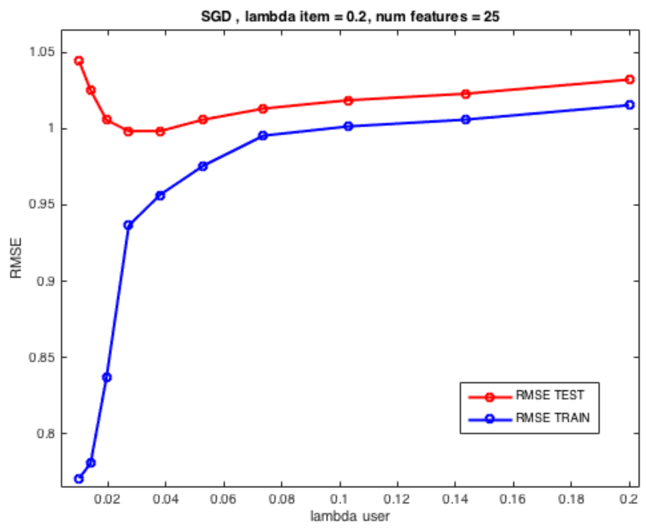

Fig. 1: SGD cross validation respect to λ_{user}

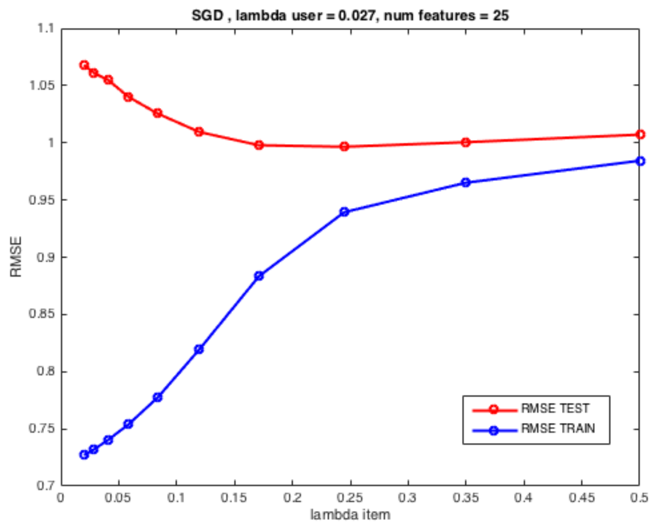

Fig. 2: SGD cross validation respect to λ_{item}

With these parameters, we achieved a RMSE equal to 0.98558 on the Kaggle platform.

C. Ensemble

Here again the tuning of the parameters resulted very important: the weights resulting from the least squares solution of the system previously described were not too far from the ones we initially guessed, but still using the computed ones improved our result.
In the table I, we summarize the scores for each method and in table II the weights we used to generate the best submission on Kaggle.

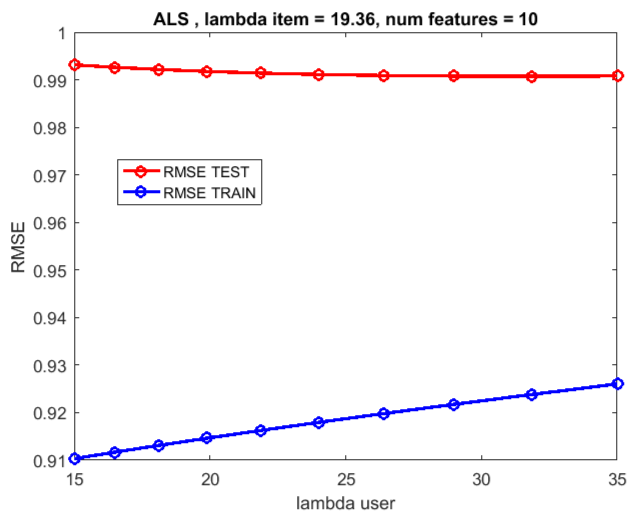

Fig. 3: ALS cross validation respect to λ_{user}

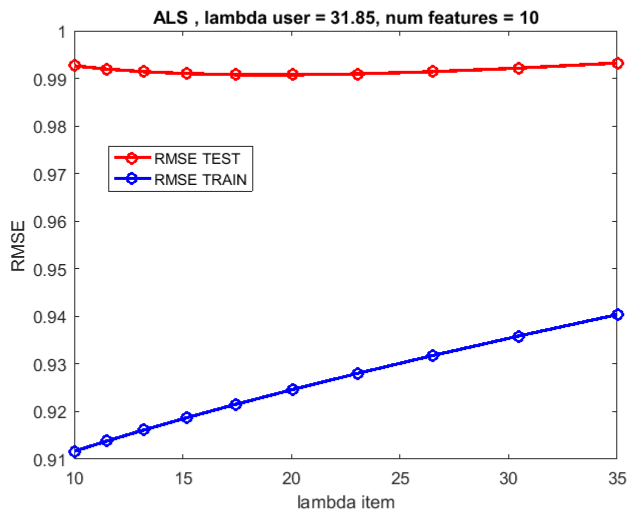

Fig. 4: ALS cross validation respect to λ_{item}

V. CONCLUSIONS

In this paper we managed to show that the combination of different methods significantly improves the final result. We achieved 0.97477 as final score on Kaggle using the combination we described. Although ALS and SGD factorize the matrix optimizing almost the same cost function, the two results are significantly different. The explanation for this fact may be that we used a different number of features, which lead to the choice of specific parameters and therefore to different models.
Possible further improvements could be adding other methods to the combination, even the same factorization method with a different number of features. Given the sensitivity of the models to the parameters, running the cross validation with a very dense refinement for each parameter can improve the model. However the drawback is the high computational time of such a process.

TABLE I: Summar of Scores

	RMSE on Kaggle
Smart average	0.99816
Global average	1.11820
Movie average	1.02982
User average	1.09268
CF item based	0.98827
CF user based	0.99935
SGD	0.97894
ALS	0.98558
Ensemble	0.97467

TABLE II: Summar of Weights

	Weights for the Ensemble
Smart average	-0.20428523411659236
Global average	0.56109367844087821
Movie average	-0.42365928091160315
User average	-0.54627855165953809
CF item based	0.37968765050773262
CF user based	0.41888801962527422
SGD	0.67664633062675772
ALS	0.15660010800838683

REFERENCES

[1] Matrix Factorizations Lesson, Machine Learning Course CS-433, Martin Jaggi and Mohammad Emtiyaz Khan 2016, EPFL

[2] Blog Post ”Netflix update: try this at home”, Brandyn Webb 2016

[3] ”Programming Collective Intelligence”, chapter 2 -making recommendation-, Toby Segaran, 2007