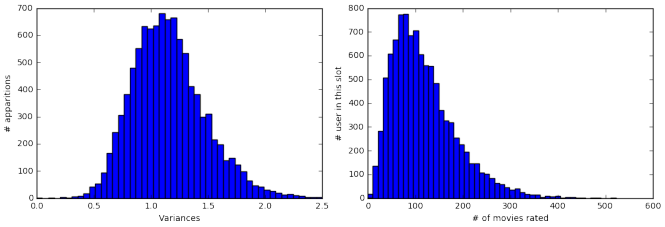
PCML CS-433: Recommender System

Gael Lederrey, SCIPER 204874, gael.lederrey@epfl.ch
Stefano Savarè, SCIPER 260960, stefano.savare@epfl.ch
Joachim Muth, SCIPER 214757, joachim.muth@epfl.ch

School of Computer and Communication Sciences, EPF Lausanne, Switzerland



(a) Distribution of variances of ratings per user. No spammers.

(b) Number of movies rated per user. Good user participation.

Figure 1: Statistical description of data

***Abstract*—Collaborative Filtering Recommender Systems for movies collection blending 12 different methods (8 direct scoring method and 4 iterative ones) in order to achieve around 0.977 RMSE on Kaggle's EPFL ML Recommender System challenge.**

I. Data description

The data represent ratings from $10'000$ users on $1'000$ movies in an integer scale from 1 to 5. This scale represent the number of *stars* given by the users, 1 being the lowest grade and 5 the best.

The training set used to train our algorithm contains $1'176'952$ ratings which represent around 12% of possible filled ratings. An other $1'176'952$ ratings are hidden from us and must be predicted by our recommender algorithm.

II. Data exploration

A. Search for spammers

One of the first step before starting learning from data is to ensure that they are real ones, and not produced by bots (spammers). As we know spammers can act in different ways: **uniform spammer** constantly rates movie in the same way, while **random spammers** randomly rates movies. In order to check their existence, we analyzed the variances of user ratings: uniform spammer would be put in evidence by null variance, while random spammer will present abnormally high variance. Figure (1a) shows the gaussian distribution of the rating variances and ensure the data are free of spammers.

B. Participation of users

Even free of spammers, data can still contains **inactive users**, i.e. users which subscribed to a plateform but never use it or never rate movies. If they are in too big number compared with active user, they can disturb learning algorithms. Figure (1b) shows histogram of number of movies rated by users and confirm us the good participation of the users.

C. User "moods" (deviation)

Because of mood/education//habits users having the same appreciation of a movie can rate it differently. Indeed, we show in figure (2) that some users systematically rate lower/higher that others. It's important to take this effect into account in both evaluation of a movie and recommendation for the user and proceed to a normalization of the ratings.

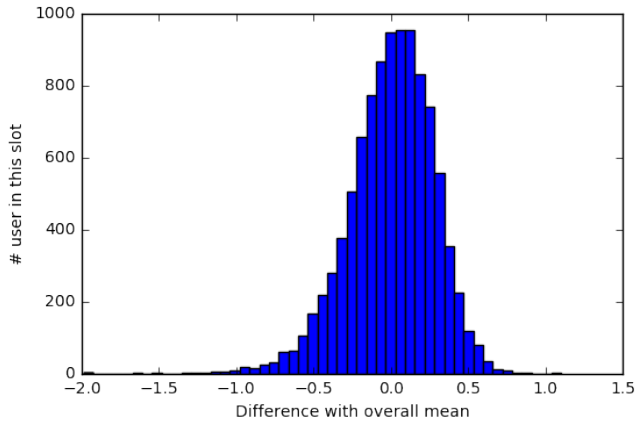


Figure 2: Difference of mean rating per user compared with overall mean.

III. Models

A. Global mean/median (2 models)

The most simple model is to take all the ratings in the train set and apply the mean or the median value. We return this value as the prediction. This give a baseline from which we can compare further model.

B. User/Movie mean/median (4 models)

Another simple model is to compute the mean or median value for the users or the movies.

C. Movie mean/median with normalized user moods (2 models)

The third set of model uses the mean or median value for each movie. We also compute the "mood" of the users this way:

$$d_u = \overline{U} - \overline{u} \quad \forall u \in U \tag{1}$$

where $\overline{U} = \frac{1}{\#U} \sum_{u \in U} \overline{u}$ and $\overline{u}$ being the average rating of the user u.

Then, we return the prediction of a user u on a movie m:

$$p_{m,u} = \overline{m} + d_u \tag{2}$$

where $\overline{m}$ is either the mean or the median of the ratings on the movie m.

D. Matrix Factorization using Stochastic Gradient Descent (MF-SGD)

Matrix factorization techniques proved to be one of the most effective strategies to implement recommender systems. Given D items, N users and the corresponding rating matrix $X \in \mathbb{R}^{D \times N}$, we aim to find two matrixes $W \in \mathbb{R}^{D \times K}$ and $Z \in \mathbb{R}^{N \times K}$ such that the quantity

$$E = \frac{1}{2} \sum_{\substack{d=1,\dots,D \\ n=1,\dots,N}} \left(x_{dn} - \left(WZ^T\right)_{dn} \right)^2 + \frac{\lambda}{2}\|W\|^2 + \frac{\lambda}{2}\|Z\|^2 \tag{3}$$

is minimized. K is a parameter, corresponding of the number of the *latent factors*; λ is a scalar that weight the regularization terms.

Different techniques have been deployed to solve this minimization problem. In this Subsection we will present the Stocastic Gradient Descent method, while in the next one we will explain the Alternating Least Square optimization. The Stochastic Gradient Descent method is a faster variant of the standard gradient descent optimization. The gradient of the functional 3 is computed only on a single element of the summation, randomly chosen. The update process then follows the same rules of the batch gradient descent. An almost certain convergence to a local minimum is guaranteed under not restrictive hypothesis.

Through a cross validation process we chosed the best values for the paramester K, λ and the number of iterations. The results that we obtained will be presented in Section V-B.

E. Matrix Factorization using Alternating Least Square (ALS)

ALS is one of the main alternatives to the SGD to solve the problem 3. It is an iterative algorithm consisting in alternately fixing one of the two matrixes W or Z, while optimizing the problem 3 with respect to the other matrix. The optimization problem at each iteration is much easier to solve compared to the one solved by the SGD. A simple least squares technique can be exploited.

For speed reasons we decided to use the open source framework Apache Spark to implement this method. Spark allows to specify the parameters K, the number of iterations and λ. We perform a cross validation to choose the best parameters, whose results will be discussed in Section V-B.

F. PyFM

We implemented two version of the SGD methods:

- An implementation from scratch, using `scipy.sparse` matrices library.
- An implementation based on the PyFM Python library, a wrapper of the C++ library libFM [1], one of the most advanced matrix factorization libraries.

PyFM is a python implementation of Factorization Machines. This library is a wrapper of the C++ library libFM [1], one of the more advanced matrix factorization libraries. It can be found on Github [2]. The idea behind the algorithm is similar to the MF-SGD III-D.

The usage of this library is really simple. It uses a few parameters:

- Number of factors corresponding to the scalar K in the MF-SGD.
- Number of iterations
- Initial learning rate

The number of iterations was fixed. The two other parameters were chosen with the use of a simple cross validation process.

G. Collaborative Filtering

This algorithm is nothing more than another implementation An other implementation of what?

IV. Blending

The *Bellkor's Pragmatic Chaos* team, winner of 2009 *Netflix Prize*, explain in its paper that its solution was obtained by blending a hundred of different models. [3] Without having the same amout of models, we proceed the same to obtain our final solution. We performe a weighted sum that we optimize using **Nelder-Mead** method provided by `scipy.optimize` library. Initial weights are set to 0.1 for each model.

V. Result

In order to create our recommender algorithm, 2 steps are required. The first step is to find the best parameters for each models. Therefore, a cross validation process was used on the models MF-SGD, ALS, Collab. Filtering and PyFM. Once the best parameters are found, the blending is applied between all the models with another cross validation process.

In this section, we first present the blending of the models as well as their parameters found after cross validation. Then, we present the benchmarks of the models and the blending.

A. Blending

Table I provides the weights after minimizing on the average RMSE. It also provides the parameters used for each model.

Model	weight	parameters
Global mean	2.87634	-
Global median	0.88256	-
User mean	-3.81181	-
User median	0.00362	-
Movie mean	-1.57271	-
Movie mean (mood norm.)	1.65276	-
Movie median	-2.27553	-
Movie median (mood norm.)	2.27933	-
MF-SGD (mood normalization)	-0.16857	$\lambda = 0.004$ features = 20 iterations = 20
ALS	0.75256	$\lambda = 0.081$ rank = 8 iterations = 24
Collab. Filtering	0.04356	$\alpha = 19$ features = 20
PyFM	0.30050	factors = 20 iterations = 100 learning rate = 0.001

Table I: Blending of models.

B. Benchmark

Table II presents the average RMSE of each model applied on the validation sets for the blending cross validation process. Last line is the result of the blending on the same validation sets.

The models were not tested individually on the Kaggle Public dataset. But the blending gives a result of 0.97788.

VI. Discussion

We can first discuss about the models. The models we used are quite simple. Therefore, using more complex algorithms instead of these would help a bit more. However with our blending method, we can use as many algorithms as we want. Indeed, since we're using an optimization algorithm, the optimization can decide itself if it should discard a model or not. However there's some risk to overfit the data. And it is happening a little bit. Indeed, the RMSE on the validations sets is a bit smaller than the RMSE on Kaggle.

Model	RMSE
Global mean	1.11906
Global median	1.12812
User mean	1.09531
User median	1.15200
Movie mean	1.03050
Movie mean (mood norm.)	0.99659
Movie median	1.09957
Movie median (mood norm.)	1.05784
MF-SGD (mood normalization)	0.99994
ALS	0.98875
Collab. Filtering	1.02776
PyFM	0.99178
blending	0.96191

Table II: Benchmark of models.

1) Why keeping model with and without normalization?: It should be legitimate to ask why are we keeping both normalized and unnormalized model for median/mean. Looking at the coefficient give a partial answer. As we see in table I, normalized and unnormalized models oppose themselves almost exactly, with a little advantage for normalized model. The effect is that the method is more taken into account for users in the tail of the mood curve than for the central ones. Then, it allows the optimizer to have finer control on the blending.

References

[1] S. Rendle, "Factorization machines with libFM," *ACM Trans. Intell. Syst. Technol.*, vol. 3, no. 3, pp. 57:1–57:22, 2012.

[2] CoreyLynch, "pyfm: Factorization machines in python," https://github.com/coreylynch/pyFM, 2016.

[3] Y. Koren, "The Bellkor Solution to the Netflix Grand Prize," 2009.