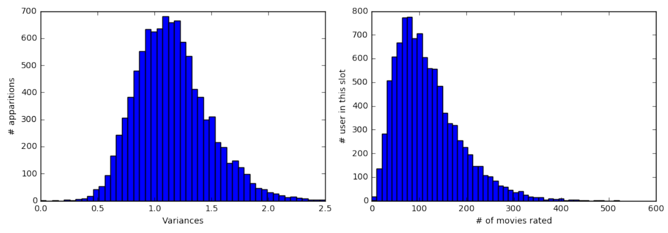
PCML CS-433: Recommender System

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*—**

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"

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.

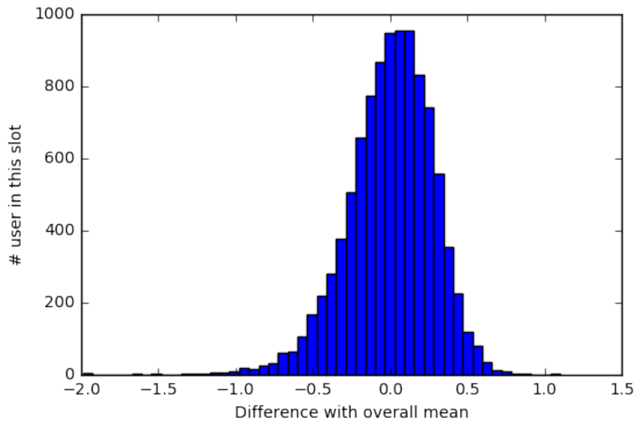


Figure 2: Difference of mean rating per user compared with overall mean.

III. Model selection

A. Models description

1) Global mean/median: 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.

2) User/Movie mean/median: Another simple model is to compute the mean or median value for the users or the movies.

3) Movie mean/median with User mood: 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.

4) Matrix Factorization using Stochastic Gradient Descent (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

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

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.

We implemented two version of the SGD methods:

- An implementation from scratch, with the help of the Scipy library to deal with sparse matrixes.
- An implementation based on the PyFM Python library, a wrapper of the C++ library libFM [1], one of the more advanced matrix factorization libraries.

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 III-B.

5) 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 III-B.

B. Models benchmark

Model	RMSE
Global mean	1.11906
Global median	1.12812
User mean	1.09520
User median	1.15163
Movie mean	1.03045
Movie mean (mood norm.)	0.99653
Movie median	1.09963
Movie median (mood norm.)	1.05793
MF-SGD (mood normalization)	0.99981
ALS	0.98861
Collab. Filtering	1.02733
PyFM	ADD VALUE

Table I: Benchmark of models.

C. 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. [2] 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 1 for each method and the final weights are listed in table (II)

IV. Result

V. Discussion

References

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

[2] Y. Koren, "The bellkor solution to the netflix grand prize," 2009.

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
		initital matrix = global_mean
ALS	0.75256	$\lambda = 0.081$
		rank = 8
		iterations = 24
Collab. Filtering	0.04356	$\alpha = 19$
		features = 20
PyFM	0.30050	ADD PARAMS

Table II: Blending of models.