PCML CS-433: Recommender System

Team: *Just keep swimming !*

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

School of Computer and Communication Sciences, EPF Lausanne, Switzerland

***Abstract*—Active Collaborative Filtering Recommender Systems for movie collection using blending of 30 different methods (8 direct scoring methods[1] and 9 iterative ones, plus 13 variations of them) in order to achieve around 0.97452 RMSE on Kaggle's EPFL ML Recommender System challenge.**

I. Introduction

Collaborative filtering is a set of techniques aiming at the creation of recommender systems. Usually, we define three types of collaborative filter: **active**, **passive** and **content-based** (the best one being obviously a mix of the three). In industry, recommenders are mainly used to suggest new item to users based on their taste: movies, music tracks, items to purchase, ...

The objective of the project is to develop a recommender system using **active collaborative filtering** (i.e. pseudonymised items[2] rated by pseudonymised users).

We first go through a general **data analysis** in order to evaluate the quality of the data (spammers and participation of the users). Then we test different models, starting from a basic mean given a prediction baseline and improving the score with more advanced algorithms.

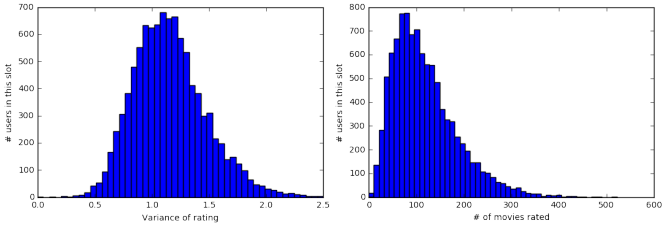
According to *BellKor's Pragmatic Chaos*[3] scientific paper [1] the best recommender is obtained through a clever blend of several models. Exploiting this approach, **we ultimately blended 30 different methods (8 direct scoring methods and 9 iterative ones, plus 13 variations of them)** eventually achieving around 0.95962 RMSE on local Cross-Validation (0.97452 on Kaggle test set).

The implemented model will be part of the **Kaggle's EPFL ML Recommender System** challenge in which predictions are rated by their RMSE compared with ground truth values. Project repository is available on GitHub [2].

[1] means, medians, ...

[2] Items anonymized by an ID, i.e. we have access neither to the name nor any content of it.

[3] Winner team of 2009 Netflix Prize



(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 the data

II. Data Description

The data represent ratings from $10'000$ users on $1'000$ movies in an integer scale from 1 to 5. Both of them are pseudonymized by an ID. This scale represents 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 filled ratings. Another $1'176'952$ ratings are hidden from us and must be predicted by our recommender algorithm in order to be scored on Kaggle platform.

III. 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 spammers** constantly rate movies in the same way, while **random spammers** randomly rate movies. 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 contain **inactive users**, i.e. users who subscribed to a platform but never use it or never rate movies. If they are in too big number compared with active users, they can disturb learning algorithms. Figure (1b) shows histograms 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 the figure (2) that some users systematically rate lower/higher than others. It's interesting to take this effect into account to create a variation of a model. (see section VII-1).

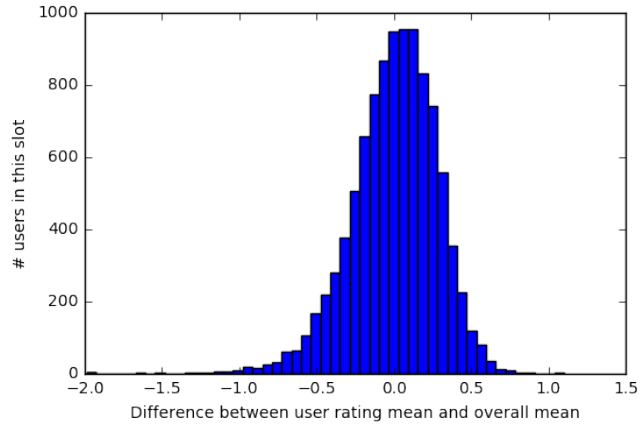


Figure 2: Difference of rating mean of each user compared with overall mean. Illustrates the differences of judgement that users give in average or what we call in this paper the "moods".

IV. Models

A. Global mean/median (2 models)

The simplest 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 score from which we can compare further models.

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)

Given D items, N users and the corresponding rating matrix $X \in \mathbb{R}^{D \times N}$, we aim to find two matrices $W \in \mathbb{R}^{D \times K}$ 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 to the number of the *latent factors*; λ is a scalar that weight the regularization terms.

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.

E. Matrix Factorization using Alternating Least Square (ALS)

ALS is one of the main alternatives to the SGD to solve problem 3. It is an iterative algorithm consisting in alternately fixing one of the two matrices 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.

PySpark[4] provides the library `pyspark.mllib.recommendation.ALS`. It is a cluster computing framework that provides to the programmers an application programming interface to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets.

F. PyFM

PyFM is a python implementation of Factorization Machines. This library is a wrapper of the C++ library libFM [3] and can be found on Github [4]. The idea behind the algorithm is similar to the MF-SGD.

[4]Apache Spark Python wrapper

G. Matrix Factorization using Ridge Regression (MF-RR)

H. Baseline

I. Slope One

J. SVD

K. KNN item/user based

V. 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 different models. [1] Without having the same number of models, we proceed the same to obtain our final solution. We perform a weighted sum that we optimize using **Sequential Least Squares Programming** (SLSQP) method provided by `scipy.optimize.minimize` library. Initial weights are set to $1/n$ for each model (n being the number of models). Instead of constraining the weights to be between 0 and 1, and to have a sum equal to 1, we choose to let the optimizer have more flexibility.

A. SLSQP method

Sequential Least Squares Programming method is a **Quasi-Newton method**. Unlike Newton method, it does not compute the Hessian matrix but estimates it by successive gradient vector analyze [5] using **Broyden-Fletcher-Goldfarb-Shanno** algorithm (BFGS). This method allows optimization for function without knowing Hessian matrix, in a short computation time.

VI. Result

In order to create our recommender algorithm, we followed the steps:

1) **Tuning the parameters** for each model one-by-one following a grid-search method scored by a 5-fold Cross-Validation.
2) **Computing the predictions** for each model for 5 folded set of data. This result to a set of $5n$ prediction tables[5] that can be tested agains 5 validation tables, providing 5-fold Cross-Validation.
3) **Optimizing the weights** for each model prediction produced by the previous step, by running a SLSQP optimization method (as explained in section V-A) on the 5 folded prediction tables.

A. Benchmark

Table I 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 blending gives a 0.97452 RMSE on Kaggle's test set.

[5] n being the number of models

Model	RMSE
Global Mean	1.11905
Global Median	1.12811
User Mean	1.09516
User Median	1.15146
Movie Mean	1.03043
Movie Mean Rescaled	1.00562
Movie Median	1.09968
Movie Median Rescaled	1.02267
Movie Median Deviation User	1.07220
Movie Median Deviation User Rescaled	1.06465
Movie Mean Deviation User	0.99661
Movie Mean Deviation User Rescaled	1.04494
MF RR	1.02774
MF RR Rescaled	1.02746
MF SGD	1.00080
MF SGD Rescaled	0.99993
ALS	0.98874
ALS Rescaled	0.98903
PyFM	0.98802
PyFM Rescaled	0.98863
Baseline	0.99925
Baseline Rescaled	1.00039
Slope One	1.00010
Slope One Rescaled	1.00032
SVD	0.99835
SVD Rescaled	0.99840
KNN Item Based	0.99031
KNN Item Based Rescaled	0.99043
KNN User Based	0.99244
KNN User Based Rescaled	0.99351
Blending	0.95962

Table I: Benchmark of models.

B. Blending

Table II provides the weights after optimization. It also provides the parameters used for each model.

VII. Discussion of the results

Excluding the trivial models based on the user/movie mean/median, we mainly focused on **Matrix Factorization algorithms**, exploiting different techniques to achieve the best factorization possible.

Blending plays an important role in our project. As we can see in table I, while models rarely score under 0.99, blending achieve around 0.96 RMSE. This is explained by the fact that certain model can compensate others for certain kind of users. The role of the optimizer is to find how to combine them in order to achieve best prediction.

1) Choice of the models: It should be legitimate to ask **why we are keeping both normalized and unnormalized model for some models**. Looking at the coefficients gives a partial answer. As we see in table II, normalized and unnormalized models oppose themselves almost exactly, with a little advantage for normalized models. The effect is that the method is more taken into account for users in the tail of the Gaussian deviation curve (figure 2) than for the central ones. Then, it allows the optimizer to have finer control on the blending.

Model	Weight	Parameters
Global Mean	1.77567	-
Global Median	1.84693	-
User Mean	-3.64246	-
User Median	0.00513	-
Movie Mean	-0.83307	-
Movie Mean Rescaled	-0.95695	-
Movie Median	-0.93869	-
Movie Median Rescaled	-0.91347	-
Movie Median Deviation User	0.93849	-
Movie Median Deviation User Rescaled	0.96461	-
Movie Mean Deviation User	1.04428	-
Movie Mean Deviation User Rescaled	0.92108	-
MF RR	0.03222	features = 20 $\lambda = 19$
MF RR Rescaled	0.03537	*(idem)*
MF SGD	-0.78708	$\gamma = 0.004$ features = 20 iter = 20
MF SGD Rescaled	0.27624	*(idem)*
ALS	0.30659	$\lambda = 0.081$ rank = 8 iter = 24
ALS Rescaled	0.31745	*(idem)*
PyFM	0.15296	features = 20 iter = 200 $\gamma = 0.001$
PyFM Rescaled	-0.02162	*(idem)*
Baseline	-0.70720	-
Baseline Rescaled	-0.56908	-
Slope One	-0.02311	-
Slope One Rescaled	0.43863	-
SVD	0.67558	$\gamma = 0.001$, $\lambda = 0.001$, iter = 30
SVD Rescaled	-0.00498	*(idem)*
KNN Item Based	-0.09500	$k = 60$, sim. metric = Pearson Baseline
KNN Item Based Rescaled	0.34178	*(idem)*
KNN User Based	0.21758	$k = 300$, sim. metric = Pearson Baseline
KNN User Based Rescaled	0.12803	*(idem)*

Table II: Blending of models. Legends: γ: learning rate, λ: regularization factor, *idem*: always refers to directly previous model

2) Overfitting: We apply several techniques to reduce as much as possible the overfitting of the models we used. In particular we used a **5-fold cross-validation both to determine the best parameters for each model and to choose the best weights in the blending.** Despite this, the model slightly overfits the data. With a 0.95962 RMSE on our local Cross-Validation and 0.97452 on Kaggle's test set, we notice a small difference that indicates overfitting.

There are many possible reasons underlying this behaviour. Probably the blending process that we used, although the proven accuracy, introduces too much complexity in the model, thus overfitting the training database.

REFERENCES

[1] Y. Koren, "The BellKor Solution to the Netflix Grand Prize," 2009.

[2] GitHub, "PCML Project Repository," 2016. [Online]. Available: https://github.com/glederrey/PCML_Netflix_and_Chill

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

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

[5] Wikipedia, "Quasi-Newton Method — Wikipedia, The Free Encyclopedia," 2016, [Online; accessed 20-December-2016]. [Online]. Available: https://en.wikipedia.org/wiki/Quasi-Newton_method