Movie Recommender System

Maryam Zakani, Zahra Farsijani, and Reza Ghanaatian
Machine Learning Course - EPFL - Switzerland

***Abstract*—In this project a recommander system for a dataset consisting of ratings of 1000 users for 10000 movies is presented. This system is designed with collaborative filtering approach in which the predication is performed according to the users past ratings. Different prediction methods are implemented based on latent factors or neighborhood models. Further, the prediction accuracy is improved by blending the better performing models. Final results show a prediction accuracy with RMSE=1.021 on the test dataset.**

I. Introduction

In the age of online platforms for shopping, providing consumers with an appropriate choice that matches their needs is a factor of crucial importance for a successful business model. The tools created for such a purpose are commonly called *recommender systems*. Such tools are particularly useful for electronic products and entertainment industry, e.g., movies and musics. Nowadays, many content providers have become interested in these tools and have made them a notable part of their online platform.

Recommender systems are built based on two approaches: the *content-filtering* approach and the *collaborative filtering* approach. The content filtering approach creates a profile for each user and each product. In this model some information is available about each product. It customizes the attributes according to third-party data such as demographic information [1] and users' preferences in the past. This approach, however, might be cumbersome especially for large datasets which there is no information about the content of some items. Collaborative filtering overcomes the difficulty by only relying on past user behavior, e.g., product ratings and predicting what users will like based on their similarity to other users. [1]. Some recent research has demonstrated that a hybrid approach, combining collaborative filtering and content-based filtering could be more effective in some cases. However, having comprehensive information about each item is not very easy. So in many cases, our project included, collaborative filtering is the only way which can be employed without any information about items.

The goal of this project is to build a recommender system using collaborative filtering approach for a dataset which consists of ratings of $D = 1000$ users for $N = 10000$ movies. We first provide a general data analysis. We then introduce several methods based on *latent factor models* and *neighbourhood models* that are used for the purpose of prediction. Finally, we improve the accuracy of the predictions by applying a clever blending [2] of several models. Final results show that we can achieve a prediction accuracy of root-mean-square error (RMSE)$= 1.021$.

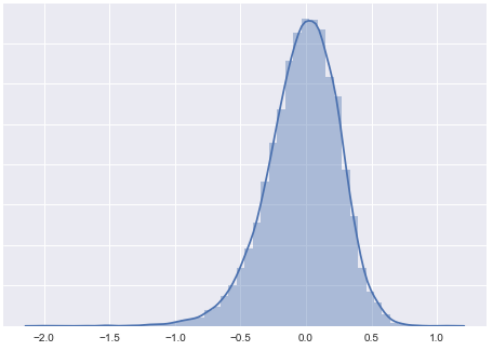

Figure 1. distribution of users mean distance from overal mean

II. Data Description

The training dataset provided in this project represents ratings of $D = 1000$ users for $N = 10000$ movies, where both users and movies are pseudonymized by an ID. The ratings are integer value from 1 to 5 which correspond to the number of *stars* given to each item (e.g. movie). The training set contains 1176952 ratings, which correspond to almost 12% of 10^7 possible ratings.

The test dataset has the same number of entries where the ratings are hidden. So the recommender system performance can be scored by uploading the predictions on CrowdAI platform.

One interesting feature of the dataset is the users mean deviation from the overall rating mean. Fig. 1 shows the distribution of the users mean distance from the overall mean, i.e., the users mean deviation. This deviation is because of the fact that different users may rate movies differently and therefore it is sometimes beneficial to take this effect into account. We will elaborate on this in Section III.

III. Models

In this section, we explain the background and the details of the models which are used to do the prediction for the ratings. The obtained results for each model are reported in TABLE I. We note that this table shows the results after optimizing the hyper parameters of each algorithm. We will elaborate on our methodology and the parameter optimization in Section VI.

A. Baseline Models

First, we look at the baseline prediction models. These models (Mean Models III-A1 and Median Models III-A3) are used in order to better capture the effects of non user-item interactions that result in various ratings; one such effect could be the symmetric bias introduced by some users, i.e. some users tend to rate higher than the others. Furthermore, some items tend to receive higher ratings from users independent of the actual interaction between the users and items. These effects can be absorbed by the baseline models as they encapsulate much of the observed data and thus help the collaborative filtering models perform better solely based results produced by user-item interactions decoupled from the non user-item interactions. This implies the importance of tuning baseline predictors for achieving better collaborative filtering results at the very end.

1) Mean Models: The simplest model for a classifier to be used as a recommender system is to take the mean value of all the ratings and suggest this value as the prediction (global mean model/user mean). However, this model only serves as a baseline for other models and has no value in reality because all movies will get similar prediction for all users. In general, recommendation systems need a history from users' preferences to be able to suggest a personalized item. So before having such data, they have to use general models such as movie mean. Movie mean can be the best model for recommendation systems before they collect user's preference.

2) Median Models: Other simple models can be obtained by taking the median value of the ratings (global median model/ Movie median/ User median).

3) Personalized Mean Movie: A smarter model will be generated by using movie mean model and considering the user general behavior in rating. In this case simply the summation of movie mean and user behavior deviation will construct the final prediction. This model performed very well in the cross validation on our train dataset. So it can be considered as a good model without any complexity in implementation.

B. Matrix Factorization with Stochastic Gradient Descent

Matrix factorization maps the user and items to a new space, i.e., latent factor space, where the interaction between the users and the items are modeled as inner product in that space and the number of latent factors shows the space dimensionality [1]. More specifically, we aim to minimize the following loss function [3]

$$L(W,Z) = \frac{1}{2}\sum_{i,j}\left(r_{i,j} - (WZ^T)_{i,j}\right)^2 + \frac{\lambda}{2}||W||^2 + \frac{\lambda}{2}||Z||^2, \tag{1}$$

where, $x_{i,j}$ are the elements of the rating matrix $X \in \mathbb{R}^{D\times N}$ $W \in \mathbb{R}^{D\times K}$ and $Z \in \mathbb{R}^{N\times K}$ are the two factorized matrices, K is the number of latent factor and λ is the regularization parameter.

In order to minimize the loss function of (1), stochastic gradient descent (SGD) can be used, where the algorithm[1] modifies the parameters in the opposite direction of the gradient with learning rate γ as

$$\begin{aligned} e_{i,j} &= r_{i,j} - (WZ^T)_{i,j}, \\ (W)_{i,:} &= (W)_{i,:} + \gamma \cdot (e_{i,j}\,(Z)_{j,:} - \lambda\,(W)_{i,:}), \\ (Z)_{j,:} &= (Z)_{j,:} + \gamma \cdot (e_{i,j}\,(W)_{i,:} - \lambda\,(Z)_{j,:}). \end{aligned} \tag{2}$$

Moreover, a normalized version of the SGD algorithm is implemented. In the normalized version, the user mean deviation is deducted before the training is performed. This deviation is added to the prediction results for each user to compensate the mean of each user.

C. Matrix Factorization with Alternating Least Squares

Because both W and Z matrices in (1) are unknown, minimizing of the cost function is not a convex problem. However, if we fix one of the matrices, it becomes convex and therefor we can compute the other. The alternating least squares (ALS) algorithm rotates between fixing one of the matrices and finding the least squares solution for the other. In this project, we use the ALS implementation by pyspark library.

The normalized version of ALS algorithm is also implemented using a similar strategy to SGD. It is worth noting that SGD implementation is easier and faster than ALS, however, ALS can be paralleled and thus it it more favorable in some cases [1].

D. Singular-Value Decomposition

The singular-value decomposition (SVD) algorithm can be used for dimension reduction of the rating matrix by selecting K number of singular values and the associated singular vectors. The decomposition can then be used for the purpose of the prediction. We use the algorithm implementation in the Python Surprise library [4].

E. pyFM

The pyFM model is a Python implementation of Factorization Machines [5]. We use the library of [6], which is a wrapper of the C++ libFM library. The general ideal of this model is close to that of SGD.

F. k-Nearest Neighbors

k-nearest neighbors (KNN) is a prediction algorithm that exploits weighted sum of all other users/items rating by using of some similarity metrics, where the more similar users/items are boosted by a larger weight. For this project

[1] https://sifter.org/simon/journal/20061211

we use Pearson baseline similarity metric and the implementation of the Surprise library [4]. The item-based KNN model was one of our best models. We also try normalization before employing this algorithm which result in the lowest RMSE in cross validation. However, this normalization seems to not be effective on the test dataset.

Table I
THE COMPARISON OF DIFFERENT MODELS

Models	RMSE CV	RMSE CrowdAI
Global mean	1.128	1.127
User mean	1.118	1.117
Item mean	1.069	1.07
Global median	1.128	1.127
User median	1.152	1.147
Item median	1.099	1.098
SGD	1.073	1.075
SGD normalized	1.109	1.103
ALS	1.031	**1.026**
ALS normalized	1.031	1.028
SVD	1.042	**1.041**
pyFM	1.028	**1.023**
kNN user based	0.996	
kNN item based	0.990	**1.025**

IV. NORMALIZATION

Since the variance between users average behaviour in rating and also the stars each movie gets are high, we should consider the potential bias in our dataset. For example, some users tend to give a higher rating even if they have similar preferences to other user in general. Therefor we can consider four different type of normalization which can be employ on this dataset.

- User normalization: Normalization will be on the average of user's ratings
- Item normalization: Normalization will be on the average of movie's ratings
- General normalization: Ignoring combination of movies and users, the average of all ratings will be considered in this case.
- Combined normalization: For each prediction, average of movie's rating and user's rating will be considered.

We employ different types of normalization described above depending on the model. For example, user normalization is used in KNN item based which was really effective by decreasing RMSE by 0.05 in the cross validation. However it does not have this effect on test set. Although doing normalization is a common approach in classification problems, it does not seem to be very effective in our project. In the end we do not take advantage of any kind of normalization in our final models.

V. BLENDING

It has been shown that a clever combination of several well-performing methods, i.e., *blending* of several models,

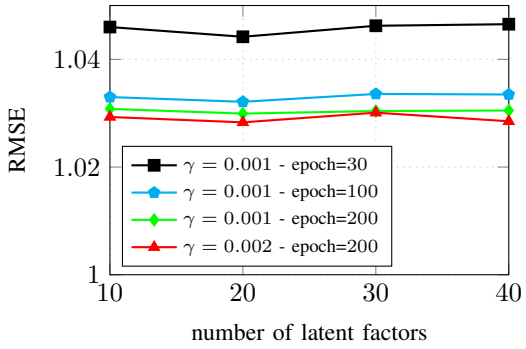

Figure 2. Parameter optimization for pyFM model

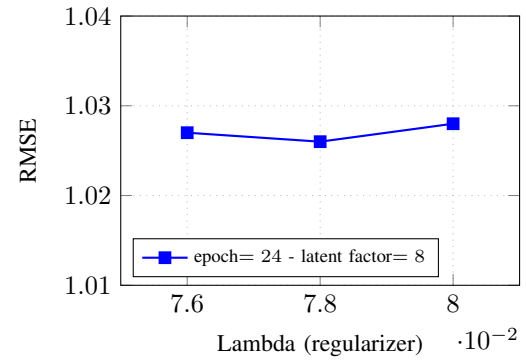

Figure 3. Lambda (regularizer) optimization for ALS model

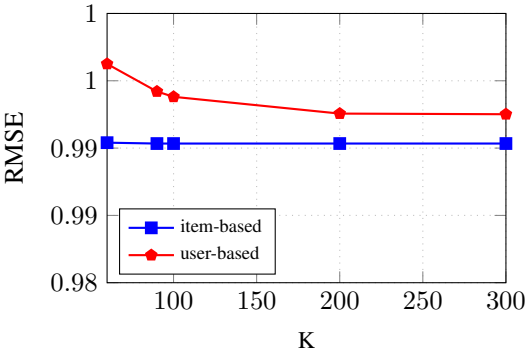

Figure 4. K (number of neighbors) optimization for KNN model

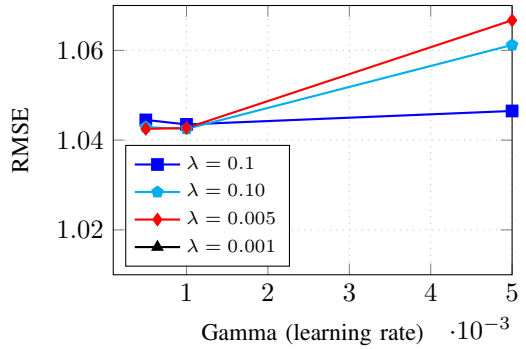

Figure 5. Gamma (learning rate) optimization for different lambda (regularizer) for SVD model

Table II
FINAL RESULTS

Models	RMSE CV	RMSE CrowdAI	Parameter	Initial Blending Weights
ALS	1.031	1.026	$\lambda = 0.076$ epochs $= 24$ factors $= 8$	0.3
pyFM	1.028	1.023	$\gamma = 0.002$ epochs $= 200$ factors $= 20$	0.3
kNN item based	0.990	1.025	$K = 90$	0.3
SVD	1.042	1.041	$\lambda = 0.001$ $\gamma = 0.01$ epochs $= 30$ factors $= 10$	0.1
Blending	-	**1.021**	-	-

can improve the prediction accuracy [2]. Such a combination can particularly be helpful when the convergence of the used algorithm to the global optimum is computationally expensive [7].

In this project we perform a weighted sum of the four better performing algorithms, i.e., ALS, pyFM, KNN, and SVD, to obtain a more accurate prediction. To this end, we start with some initial weights which are chosen according to the performance of each algorithm in a way that sum of all the weights is unit. We then use sequential least squares programming (SLSQP) methods from scipy library to optimize the weights.

VI. RESULTS AND DISCUSSION

A. methodology

In order to build our recommender system, we first explore the prediction errors by different models. To this end, we perform a 5-fold cross-validation on the training dataset. The hyper parameters for each model are also optimized by a greed-search method. Fig 2, Fig. 3, Fig. 4, and Fig. 5 show the RMSE during the parameter optimization for pyFM, ALS, KNN and SVD algorithms, respectively. Once the optimum parameter values are derived, we evaluate the prediction on the test dataset by CrowdAI submission results.

B. Final Results

TABLE II shows the final results for the four best models and the optimized values of their parameters. These four models are finally selected for blending and the best blending weights for each algorithm are reported in this table. We note that since the normalized version of SGD and ALS algorithm could not achieve a better prediction performance compared to their standard version, we do not explore the normalization for other algorithm of TABLE I and we only use the standard algorithms for blending.

VII. CONCLUSION

A movie recommander system using collaborative filtering approach was implemented. Different models with various predication accuracies were implemented. The final implementation achived a prediction accuracy of RMSE$= 1.021$ by combining the results of four better performing algorithms, i.e., pyFM, ALS. KNN, and SVD.

REFERENCES

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[2] Y. Koren, "The bellkor solution to the netflix grand prize," *Netflix prize documentation*, vol. 81, pp. 1–10, 2009.

[3] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in *Advances in neural information processing systems*, 2008, pp. 1257–1264.

[4] "Surprise, a simple recommender system library for python," *Available: http://surpriselib.com/*, 2016.

[5] S. Rendle, "Factorization machines with libfm," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 3, no. 3, p. 57, 2012.

[6] CoreyLynch, "Factorization machines in python," *Available: https://github.com/coreylynch/pyFM*, 2016.

[7] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.