Project 2. Recommender System

Cho Hyun Ji, Poopalasingam Kirusanth, Rodriguez Natalia

***Abstract*—In the following project, our goal was to build a recommender system to suggest items, such as movies, to users. For this purpose, we implement and optimize a collaborative filtering algorithm using several models. Our final model was optimized using blending an considering six different models: *BaselineOnly*, *SVD*, *SlopeOne*, *KNNBaseline*, *KNNBasic* and *NMF*, we obtained a final RMSE of 1.033 in the CrowdAi platform.**

I. Introduction

In this project, we implement a recommender system based on collaborative filtering in order to predict ratings of movies. A system like this has high significance due to its applicability in recommending movies that may be highly enjoyed by a given user. In section II we started by exploring the raw data and deciding how it can be wrangled for our purposes. Then, in section III, we describe the different implementations of matrix factorization, the manipulation we did to the data, and the different methods we used to analyze the data set. Furthermore, in section IV we explain how we chose each of the models we considered for our final configuration, we based our analysis in single model performance and Pearson correlation coefficient. Finally, in section VI we discuss other methodologies that could be have been implemented to improve the overall performance of the model.

II. Data Description

The data set contains ratings from 1 to 5 of 1,000 movies by 10,000 users. The training set consists of 1,176,952 ratings, and a second set of 1,176,952 ratings is hidden from us and needs to be predicted by our recommender algorithm for the CrowdAI competition.

Every user is identified with a given ID and the number of ratings per user varies from 3 to 522. As we can observe in Figure 1 the number of ratings per user varies considerably; therefore, this could lead to non-uniform accuracy in the predictions across users. Furthermore, it is important to take into account that the number of ratings per movie varies considerably, as shown in Figure 2a. The minimum number of ratings per movie was 8 and the maximum was 4590; therefore, it is possible that more accurate predictions could be made for movies with a higher number of ratings than for movies with a low number of user ratings.

Finally, to asses if the data was heavily influenced by spammers we determined the variance in the ratings provided by each user. As we can see in Figure 2b, the variance in the ratings per user is close to a Gaussian distribution, which leads us to conclude that the data set is not strongly influenced by spammers.

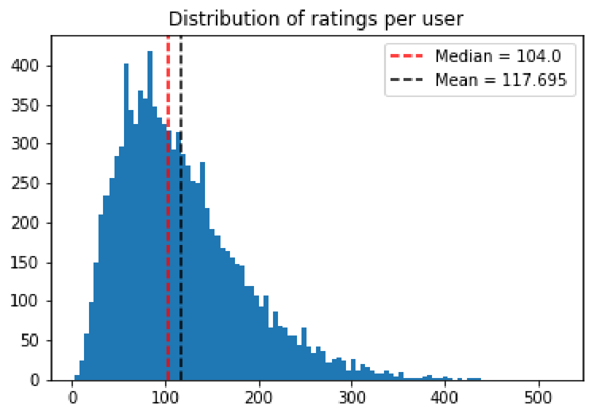
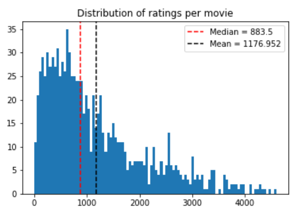
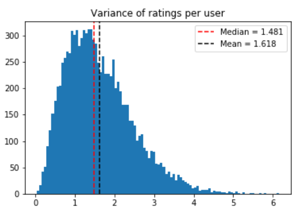


Fig. 1: Distribution of ratings per user



(a)



(b)

Fig. 2: Data Analysis

III. Prediction methods

In the following section, we will briefly present the models that we used throughout our analysis. Given a matrix X, $X_{d:}$ represents the d_{th} row of X and $X_{:n}$ represents the n_{th} column of X. Furthermore, Ω is the set of all non-zero ratings in the training set and we labeled each item with the indices (d, n) corresponding to each user and movie. Following this notation, the global mean $\hat{X}$, the user mean $\hat{X}_n$ and the movie mean $\hat{X}_d$ are given by:

$$\hat{X} = \frac{1}{|\Omega|} \sum_{(d,n)\in\Omega} X_{d,n} \quad (1)$$

$$\hat{X}_n = \frac{1}{|\Omega_{:n}|} \sum_{d\in\Omega_{:n}} X_{d,n} \quad (2)$$

$$\hat{X}_d = \frac{1}{|\Omega_{d:}|} \sum_{n\in\Omega_{d:}} X_{d,n} \quad (3)$$

- *Matrix Factorization*: Given matrix X we want to find W and Z such that:

$$X \approx WZ^T \tag{4}$$

We solve this as an optimization problem using the following algorithms:

– *Stochastic Gradient Descent (MF-SGD)*: Minimize the following: (8).

$$\min_{(W,Z)} L(W,Z) = \frac{1}{|\Omega|} \sum_{(d,n)\in\Omega} \underbrace{\frac{1}{2}[X_{dn} - (WZ^T)_{dn}]^2}_{f_{dn}} + \frac{\lambda_{user}}{2}\|Z\|^2 + \frac{\lambda_{movie}}{2}\|W\|^2 \tag{5}$$

where λ_{user} and λ_{movie} are the regularization parameters. We determined λ_{user} and λ_{movie} using a grid search(check!). On the other hand the expressions for the updates of W and Z are given by:

$$W^{t+1} = W^t + \gamma \nabla_W f_{dn} \tag{6}$$
$$Z^{t+1} = Z^t + \gamma \nabla_Z f_{dn} \tag{7}$$

where γ represents the learning rate.taking into account the global mean and the user mean, i.e, evaluating $X = \hat{X} + WZ^T$ and $X = \hat{X}_n + WZ^T$ respectively.

– *Alternating Least Square (MF-ALS)*: With the *MF-ALS* method we aimed to solve the same minimization problem (5) by fixing one of the matrices W or Z while alternating the other. This method is an alternative to *MF-SGD* and allows to implement an easier parallel update on matrices W and Z. We can take into account the global mean and the item mean, i.e, considering $X = \hat{X} + WZ^T$ and $X = \hat{X}_d + WZ^T$ in the minimization problem (5).

- *k Nearest Neighbors (k-NN)*: The *k-NN* method is an algorithm that performs a prediction based on the weighted sum of the k nearest neighbors ratings. In the Surprise library V-A, the *k-NN* method is implemented using the following relation:

$$\hat{r}_{dn} = \frac{\sum_{\nu\in\Omega} \text{sim}(d,\nu)\cdot r_{\nu n}}{\sum_{\nu\in\Omega} \text{sim}(d,\nu)} \tag{8}$$

where $\hat{r}_{dn}$ represents the prediction and the $\text{sim}(u,\nu)$ function is a dictionary of options for the similarity measure. We used the Pearson baseline as similarity measure and we set $k = 150$.

– *k Nearest Neighbors with mean*: follows the same relation than the *k-NN* model presented above but takes into account the global mean in the prediction relation (8).

- *Baseline*: This determines a baseline estimate per user and per movie using the bias of each of them and the global mean. It was implemented from the Surprise libraryV-A and it is given by the relation:

$$\hat{r}_{dn} = \hat{X} + b_d + b_n \tag{9}$$

where b_d represents the bias per user and b_n represents the bias per movie.

- *Single Value Decomposition (SVD)*: This method makes a prediction for a user rating that follows the relation:

$$\hat{r}_{dn} = \hat{X} + b_d + b_n + q_n^T p_d \tag{10}$$

with q_n the user feature matrix and p_d the movie feature matrix.

- *Slope One*: This method follows:

$$\hat{r}_{dn} = \hat{X}_n + \frac{1}{|R_n(d)|} \sum_{j\in R_n(d)} \text{dev}(n,j) \tag{11}$$

where $R_n(d)$ is the set of j items rated by user d that have atl least one common user with n. The function $\text{dev}(n,j)$ is defined as the average difference between the ratings of items n and j.

- *Non-Negative Matrix Factorization (NMF)*: This method follows the same matrix decomposition that we presented in (4) but imposes the condition that the three matrices must have non-negative elements. Given that our original matrix only contains non negative elements it was possible to implement *NMF*. This method was also implemented with the Python Surprise library V-A.

IV. Results

We used two factors to asses the performance of each model. First, we measured the individual performance of each model by the obtained RMSE. Second, we used the Pearson correlation coefficient to evaluate the similarity of the contributions between each pair of models.

A. Individual Performance

We established a threshold of 1.015 for the RMSE and drop the models with higher RMSE. In Figure 3, we observe that this leads us to drop Matrix Factorization with *SGD*, Matrix Factorization with *ALS*, and the models that only considered the global mean, the user mean and the movie mean.

From Figure 4, we observe that given our threshold we would only keep *BaselineOnly*, *CoClustering*, *SlopeOne*,

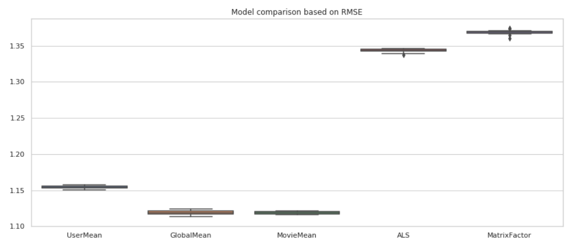


Fig. 3: Model Comparison (a)

KNNBaseline and *NMF*; however, we decided to also keep *SVG* and *KNNBBasic* given their Pearson correlation coefficient, we will explain our decision in more detail in the following section.

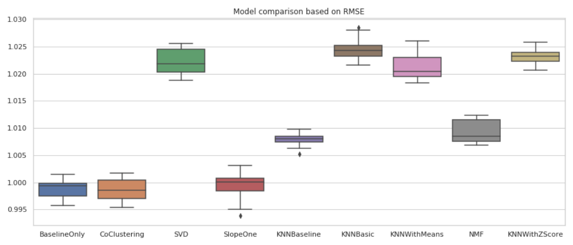
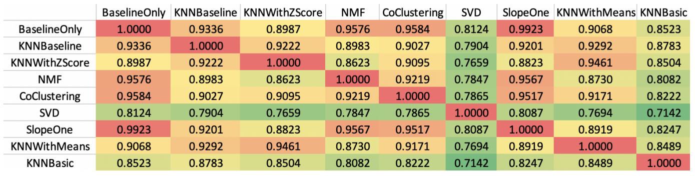


Fig. 4: Model Comparison (b)

B. *Pearson Correlation Coefficient*

Given that the Pearson correlation coefficient between the *BaselineOnly* and the *CoClustering* models is very high we decided to keep just one of the two, the kept one is the *BaselineOnly model*. Furthermore, since the *KNNBasic* model, the *KNNWithMeans* and the *KNNWithZScore* also showed high correlation between each other we decided to keep only one of them. We kept the *KNNBasic* model given that it is less correlated with the remaining models as seen in 5.



	BaselineOnly	KNNBaseline	KNNWithZScore	NMF	CoClustering	SVD	SlopeOne	KNNWithMeans	KNNBasic
BaselineOnly	1.0000	0.9336	0.8987	0.9576	0.9584	0.8124	0.9923	0.9068	0.8523
KNNBaseline	0.9336	1.0000	0.9222	0.8983	0.9027	0.7904	0.9201	0.9292	0.8783
KNNWithZScore	0.8987	0.9222	1.0000	0.8623	0.9095	0.7659	0.8823	0.9461	0.8504
NMF	0.9576	0.8983	0.8623	1.0000	0.9219	0.7847	0.9567	0.8730	0.8082
CoClustering	0.9584	0.9027	0.9095	0.9219	1.0000	0.7865	0.9517	0.9171	0.8222
SVD	0.8124	0.7904	0.7659	0.7847	0.7865	1.0000	0.8087	0.7694	0.7142
SlopeOne	0.9923	0.9201	0.8823	0.9567	0.9517	0.8087	1.0000	0.8919	0.8247
KNNWithMeans	0.9068	0.9292	0.9461	0.8730	0.9171	0.7694	0.8919	1.0000	0.8489
KNNBasic	0.8523	0.8783	0.8504	0.8082	0.8222	0.7142	0.8247	0.8489	1.0000

Fig. 5: Pearson Correlation Coefficient

C. *Parameter search*

We performed grid search on different models separately in order to check if the prediction can be improved by choosing better parameters. The only considerable improvement was by testing for different k-neighbours which is shown in 6. The default value chosen by the Surprise library V-A is 40. Increasing it to 150 improved our performance from 1.022839 to 1.014130

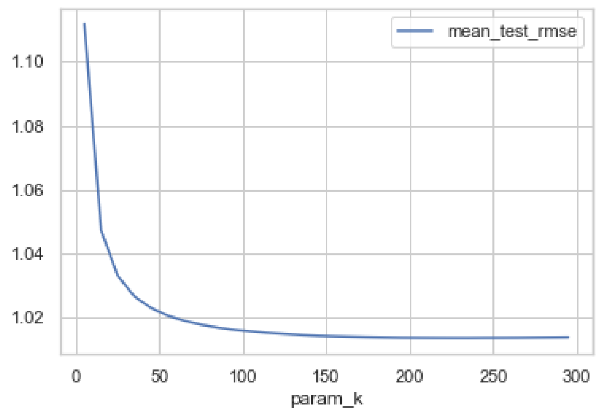


Fig. 6: Grid search on KNN over different k-Neighbours

D. *Blending*

We proceeded with our analysis with the six models that presented better results, i.e., *BaselineOnly*, *SVD*, *SlopeOne*, *KNNBaseline*, *KNNBasic* and *NMF*. To improve the performance of our general model we blended this six models and determined the best weights for each of the models by using the minimize-function provided by Scipy V-B.

The final weights were chosen using cross validation. Combining the six different models performed overall better than each model individually.

E. *Model comparison*

In the table bellow we present the obtained individual performance for each model and how blending improve the performance of the model as was measurny the RMSE.

Model	RMSE	Blending Weights
BaseLineOnly	0.9958	-0.7855
SVD	1.0187	0.2610
SlopeOne	0.9938	0.9528
KNNBaseLine	1.0052	0.4754
KNNBasic	1.0216	0.1361
NMF	1.0068	-0.0333
Final model	0.9882	

TABLE I: Model Comparison

V. Technical Details

A. *Surprise*

In order to build our recommender systems, we used Surprise, a python scikit library, which provides several prediction algorithms, neighborhood methods, and matrix factorization-based methods.

B. *Scipy*

In order to find the minimum weights for the blending of the different models we used the minimize function provided by Scipy.

VI. Discussion

In order to get a robust prediction, we started by evaluating the performance of each model individually, but the results were poor or the parameters were not optimized for that model. We tried to combine several models by averaging their predictions, which increased the performance slightly. However, in the end, we decided to use blending in order to give each model different weights, including negative weights.

As we explained before, for our final model, we selected six models I based on their individual performance given by their RMSE and the Pearson correlation coefficient. We used a blending of these models following the strategy proposed in [1] using the scipy minimization function V-B. Using our final configuration of models and weight we obtained an RMSE of 1.033 on the CrowdAi platform.

A future improvement of our approach would be to include more models, also poor-performance ones, like]textitALS or the global mean. Given this poor performance, the hope is that blending will assign a lower weight if the overall performance can be improved by that, best case these models get zero weights assigned.

In future work, we could implement a splitting of the data set. With this, we could reduce the contribution of large outliers and improve the predictability across both user and movies based on the amount of available ratings. Additionally, we could implement the *SDVpp* method provided by the Surprise library V-A in the final blending. We attempted this method; however, the training duration was considerably longer compared to *SVD*. Furthermore, the splitting of the data can be combined with blending.

References

[1] R. M. Bell, Y. Koren, and C. Volinsky, "*The Belkor solution to the Netflix Prize*," 2007.