Machine Learning-Project 2: Recommender System

Pierre Fouche, Matthias Leroy, Alexandre Poussard
MSc Data Science - EPFL, Switzerland

***Abstract*—The aim of this project was to create a recommender system for movies. To do so we predicted the ratings of users on movies based on a dataset of ratings. Our model was based on ones such as Stochastic Gradient Descent, Alterning Least Squares, K-Nearest-Neighbors and some others. We decided to merge them all in order to obtain the best predictions.**

I. Data

The data consists of ratings of different movies by different users. There are 10000 users and 1000 movies. The dataset contains only 1 176 192 ratings over the 10 000 000 possibilities, which means that we had to deal with a 88.2 % sparse matrix. Here we did not do any preprocessing, we kept the data as such. The metric we used to measure our models is the Root Mean Square Error (RMSE).

II. Models

We started by reading a lot of papers about recommender systems before starting the project including the paper of the Netflix Prize[1] winning solution. The best solution was obtained by mixing different models, this is why we decided to do the same.

First we worked on different models individually. We started by training each of these models and tried to find the best hyperparameters giving the lowest RMSE. We first worked on some models seen in class (ALS, SGD) and then discovered a python library called Surprise[2]. Suprise provides different methods to predict the ratings, we tested some of them.

For training, we splitted our dataset in two sets: the first one containing 90% of the data, the training set and the other one 10% of the data, the test set.

A. Alterning Least Squares (ALS) with Non negative factorization

We started by the ALS algorithm seen in class. So we first obtain the W and Z of the factorization: $R \approx W \cdot Z^T$ where R is $N \times M$ matrix, W is a $N \times K$ matrix and Z a $M \times K$ matrix. K is the number of features. To get these two matrices we decided to use the Non negative matrix factorization provided by sklearn[3]. We then proceeded to update the two matrices up until convergence. In order to obtain the lowest RMSE we tuned three parameters: the number of features k, the λ_{user} and the λ_{item}. We found an error of 0.994 on our test set. As you can see at Fig.1 the best k was 3. For the λ we chose $\lambda_{user} = 0.2$ and $\lambda_{item} = 0.9$. With this model our score on Kaggle was 0.98914. Knowing that we had to train some other models we decided to stop spending time on this one.

Figure 1. RMSE of ALS for different number of features.

B. Baseline

This algorithm, implemented by Surprise[2], predicts the ratings with the formula: $\hat{r}_{ui} = \mu + b_i + b_u$ where μ is the mean, b_i is the item bias and b_u is the user bias. With this method we achieved a score of 1.0012 on our test set.

C. Stochastic Descent Gradient(SGD) with Non negative factorization

Doing the same as for ALS, we started by factorizing our ratings matrix with the Non negative factorization. We get W and Z as factorization, the goal here is to compute the gradient for W and Z. Then we computed the updated Z: $Z = Z - \gamma \nabla Z$ and the updated W: $W = W - \gamma \nabla W$. And we repeat this until convergence. Once again to obtain the best result we had to find the best values for the parameters γ, k, λ_{item} and λ_{user}. We got an RMSE of 0.9837 on our test set with k=20, $\lambda_{user} = 0.01$ and $\lambda_{item} = 0.016$ and $\gamma = 0.025$. We can see on Fig. 2 that the number of features has little influence on the RMSE. On Kaggle we obtained an error of 0.97833.

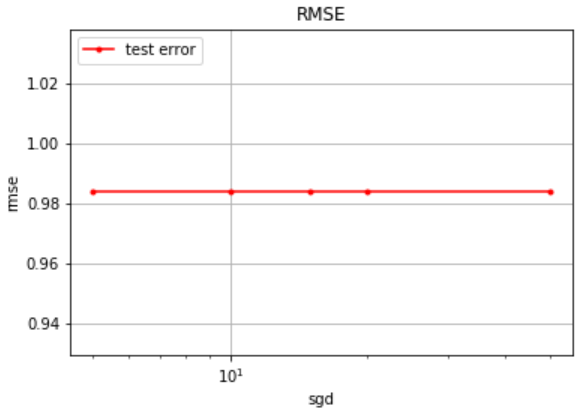

Figure 2. RMSE of SGD for different number of features.

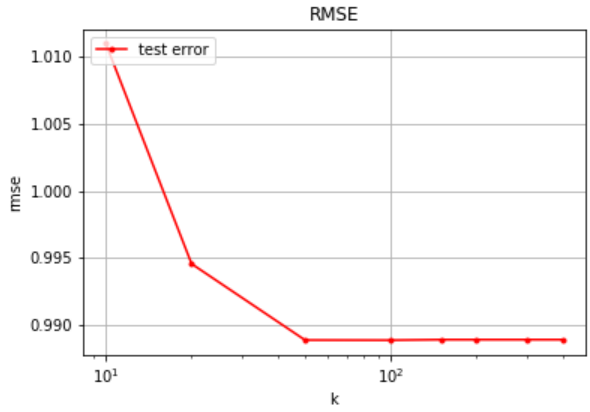

Figure 3. RMSE of KNN for different k.

D. Global mean

This method consists to take the mean of the entire matrix and predict this value for the missing entries. With this method we obtain a RMSE of 1.1215 on the test set.

E. User mean

We took the mean of ratings by user and predict this value for each user. We had an error of 1.033 on our test set.

F. Item mean

Same as before but with the items. We obtained an error of 1.096 on our test set.

G. Slope One

Here we used the Surprise[2] method: Slop One. This is a simple collaborative filtering algorithm. The prediction is set as:

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{i \in R_i(u)} dev(i,j) \tag{1}$$

where $R_i(u)$ is the set of relevant items, i.e. the set of items j rated by u that also have at least one common user with i. $dev(i,j)$ is defined as the average difference between the ratings of i and those of j:

$$dev(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - r_{uj} \tag{2}$$

Our error with Slope One was 1.0018 on the test set.

H. KNN

Onto K-NN now, which can be user based or item based, we applied only the item-based one as the other one was taking too much time. This algorithm computes the rating of an item by summing the weighted ratings of the other items. The weights are the similarities between items. As we can see on Fig. 3, the best numbers of neighbors k was 100. Our error with this method was 0.9891.

I. Singular Value Decomposition

Let's try the SVD algorithm implemented by Surprise[2]. The prediction $\hat{r}_{ui}$ is set as: $\hat{r}_{ui} = \mu + b_u + bi + q_i^T p_u$ where b_u and b_i are the bias and $R \approx Q^T P$. As shown by Fig. 4 we obtained an error of 1.0022 with 10 features.

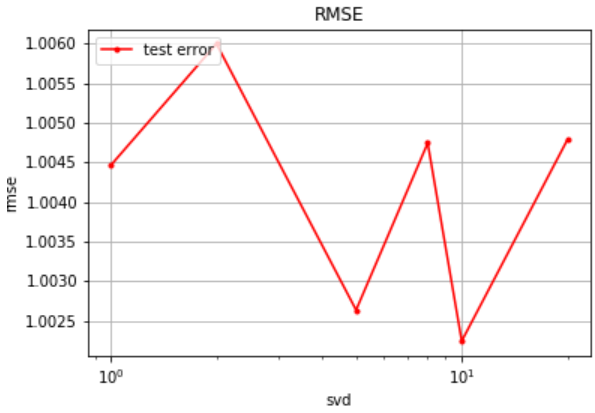

Figure 4. RMSE of SVD for different numbers of features.

III. Mixing our models

Once all of our models were trained and optimized, we tried to mix them. To do so, our main idea was to weight every prediction we obtained and add them together. Then we computed the RMSE of this mixed prediction. So we had to find a way to minimize the error by choosing the right weights. Fortunately for us the function scipy.optimize.minimize[4] does this exactly. We also wondered if we should add a constraint on the weights or not. We started by initializing the weights to $\frac{1}{n_{Models}}$ and our first option was to add the constraint: $\sum_{i=0}^{n} w_i = 1$. The other option was to add no constraint on the weights. After

testing on our test set we chose the second option which gave us the lowest error.

Models	Weights
ALS	0.155485
SGD	0.6606293
Global mean	0.04616225
User mean	0.08127007
Item mean	- 0.06675145
SVD	0.04531555
Baseline	- 0.28565113
KNN	0.40258731
SlopeOne	- 0.02940758

Table I
WEIGHTS OF THE DIFFERENT MODELS

With these weights our error on the test set was 0.97832. On Kaggle we obtained the first place with 0.97317.

IV. CONCLUSION

The best result we obtained was with our mix of models. This score gave us the first place on Kaggle. We noticed that the mentioned mix of models gave us a lower error than every single one of the other models. In a sense, the mix takes only the best out of each model. With more time and more computation power we think that we could achieve a better score by doing a very deep optimization of every model and by adding more models.

REFERENCES

[1] Y. Koren, "The bellkor solution to the netflix grand prize," *https://www.netflixprize.com/assets/GrandPrize2009_-BPC_BellKor.pdf*, 2009.

[2] N. Hug, "Surprise," *https://github.com/NicolasHug/Surprise*, 2017.

[3] *http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html*.

[4] "https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize."