CS-433 Machine Learning - Project 2 Recommender system

Alix Jeannerot [1], Nicolas Lefebure [2], Samuel Dubuis [1]
[1]*School of Computer and Communication Sciences, EPF Lausanne, Switzerland*
[2]*School of Mechanical Engineering, EPF Lausanne, Switzerland*

***Abstract*—This project presents a machine learning model that aims to predict item recommendations to a database of users. Based on the ratings of several items on a scale from 1 to 5 (no additional information about the movies or users), we combine different prediction algorithms in order to guess such recommendations.**

I. Introduction

We develop our machine learning model in two distinct stages. In the first place, we train a series of prediction algorithms and optimize the parameters of each one in order to minimise the RMSE of the predicted ratings over the test set. This part uses the library *Surprise* [1] which is a Python scikit building and analyzing recommender systems set of tools. In the second place, we combine all the algorithms together with a ridge regression model and optimize the weighting parameters to obtain the best linear combination. This part uses the library Scikit-learn [2], which is a free and open source Python library for machine learning.

II. Problem set up

A. Analysing the Data Set

As we have acquired ratings of 10000 users for 1000 different items, the training set is a matrix of ten millions components where lines represent users and columns represents items. Each rating of a user is given by a specific id. For instance, r61_c1,3 is saying that the user 61 gave a rating of 3 to the item 1. If a component of the matrix is null, it means that the user did not mark the item. It is important to notice that the matrix is filled with null values as we only have 1176952 ratings. It means that 88.2 % of the matrix is made of zero values. The use of a compressed version of the matrix in a Pandas Dataframe has considerably helped to increase the speed of learning algorithms. The ratings are distributed as shown in Fig.1

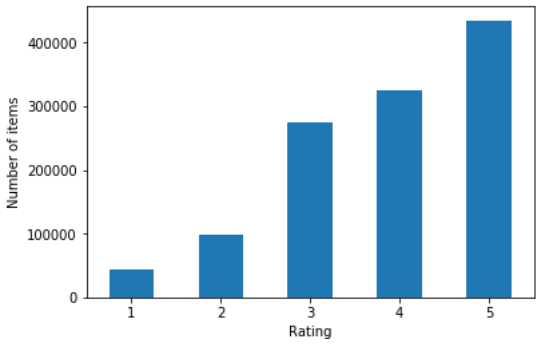

Fig. 1: Distribution of the five ratings

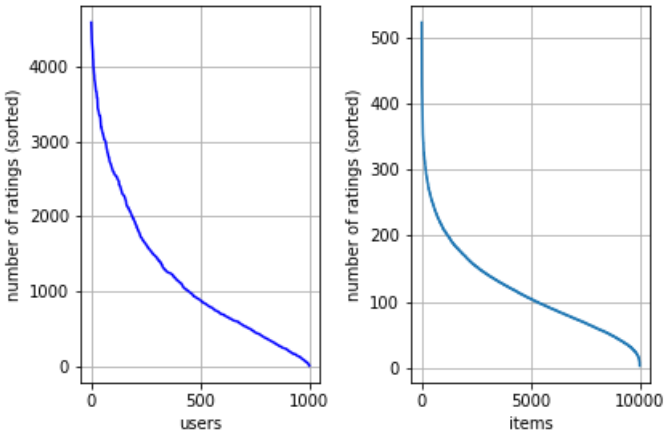

Fig. 2: Left plot: the (ordered) number of ratings for each user. Right plot: the (ordered) number of ratings for the movies.

B. Cleaning the Dataset

The database is very simple. When trying to eliminate some potential outliers, we did not find a significant number of items that have a too small number of ratings to be considered in our model. For example there is only 139 items that are rated less than 10 times. Given the size of our data-set this number is too small to make a significant change in our results. The figure 2 shows the number of rating for each user and each item. Both for users and items, we can see that the number of rating increases quickly, meaning that there is no need to reject values.

C. Performance criteria

The performance of all the filtering algorithm that we develop is measured thanks to the prediction error, here given by the RMSE (Root Mean Square Error). Its equation is given by the following:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} \left(y_{\text{pred},i} - y_i\right)^2}{n}}$$

where y_{pred} are the predictions, y are the actual or test values and n is the size of the dataset.

III. Stage 1: Prediction algorithms

A. Algorithms used

This section presents the several Surprise algorithms used for the purpose of this project. To embrace all the possibilities offered by the library, all the families of predictions algorithms were used, except for the Normal predictor who was not used because it basically makes random choices and therefore would product "noise" among our results for the second stage in section IV. Additional information on these algorithms can be found in reference [1]. It is important to state that all algorithms comply to the same notation. The letter u represent user while letter i represent items. In this manner, $\hat{r}_{ui}$ is the prediction of the rating r_{ui} made by user u to item i.

1) Baseline: This algorithm predicts the baseline estimate for given users and items. The baseline is defined as the following:

$$\hat{\boldsymbol{r}}_{ui} = b_{ui} = \mu + b_u + b_i$$

Then the following regularized cost function is minimized:

$$\sum_{r_{ui} \in R} (r_{ui} - b_{ui})^2 + \lambda \left(b_u^2 + b_i^2\right)$$

In surprise, the optimisation of this function can be perform by two method: Stochastic Gradient Descent or Alternating Least Squares method. Both of them were tested and the second one exhibited better results.

2) Co-clustering: It is a collaborative filtering algorithm based on co-clustering. Thanks to a straightforward optimization method, the algorithm assigns cluster (C_u, C_i) and co-clusters (C_{ui}) to the users and items. By averaging these clusters $(\overline{C_u}, \overline{C_i}$ and $\overline{C_{ui}})$, the prediction is given by:

$$\hat{r}_{ui} = \overline{C_{ui}} + \left(\mu_u - \overline{C_u}\right) + \left(\mu_i - \overline{C_i}\right)$$

In the scope of this project, we tuned the parameters: number of clusters for items and users and the number of epochs (corresponding to the number of iteration for the optimization).

3) k-NN inspired algorithms: The aim of this algorithm is to make a prediction accordingly to predicted ratings for items or user close (considered as "neighbors") to this prediction. In this case, the prediction law is given by the following:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

Here, the set $N_i^k(u)$ represents the k neighbors of the users. Also, $\text{sim}(u, v)$ is a function that characterises the similitude between two parameters: in this case, the function that described the similarity between two users was the MSD (Mean Squared Difference). This algorithm correspond to the basic k-NN. Surprise offers the possibility to use other algorithms derived from this one. In particular, we used a model (k-NN Means) that could take into account the mean ratings of each user by centering the predictions around this mean. The prediction is given by:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

An other version was used: k-NN Baseline. Rather than rectifying with the mean ratings of users, we correct with a similarity measure. The prediction is then:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

4) Slope One: Slope One is an algorithm that is also used for collaborative filtering. It is particular in the way that the predictor is linear with no coefficient (hence "slope one"), which is a pretty simple approach to standard linear regression.

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

where $\text{dev}(i, j)$ is given by:

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

As $R_i(u)$ 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. And $\text{dev}(i, j)$ is defined as the average difference between the ratings i and those of j. This algorithm has the advantage of being simple and not having to define hyper-parameters.

5) Matrix Factorization-based algorithms (SVD & NMF): Matrix factorization algorithms basically work by decomposing the user-item interaction matrix into a product of two distinct matrices. These two matrices represent respectively users and items in a lower dimensional latent space. It means that that the rows or column of the new matrices now represent a "latent factor" that can be tuned (thanks to grid search in our case). These factors can be seen as classifier in categories for the users of items. Again, the prediction law is given by the following:

$$\hat{\boldsymbol{r}}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

Also, bias can be integrated in the model. In this case μ is the global bias while b_u and b_i logically represent bias for users and items. The values q_i and p_u are the so called factors.

We use two Matrix Factorization-based algorithms: SVD and NMF. The first one minimizes the following regularized squared error:

$$\sum_{r_{\text{we}} \in R_{\text{train}}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2\right)$$

The hyper-parameters of the algorithm are much more various than the previous ones: the number of factors, the number of epochs, bias parameters (to be included or not), the learning rate γ, the regularization term λ, the mean and the

standard deviation for the initial distribution for factor vectors initialization. The second algorithm (NMF) is similar to SVD. They differ in the way that the user and item factors are kept positive and that the distribution of initialization is uniform.

B. Grid Search method

For each algorithms, we start by performing a 5 fold cross-validation and then a 3 fold cross-validation to accelerate calculations. We then determine the hyper-parameters that give the lowest RMSE thanks to grid search. Carry out all these calculations is very time-consuming, so most of the time, work is done in parallel by running "jobs" simultaneously (a "job" is a possibility of parameter during a cross validation). However, this method is not used with k-NN because segfaults appeared when more than two jobs are running at the same time. We also had memory space problems from time to time. In order to solve this problem, each time after performing the grid search and fitting the algorithm on 70% of the total dataset, we dumped the model locally, deleted the variables and reloaded them at the end to be able to use them later. The TABLE I presents the range that was used for each hyper-parameter for all the algorithms during the grid search. For k-NN algorithms, k is the maximum number of neighbors to take into account for aggregation while k_min is the minimum. sim_options is a Boolean number which states whether similarities will be computed between users or between items. As it has a huge impact on the performance, we tested as much as possible similarities but most frequently, the best one was MSD. For Matrix Factorization-based algorithms, n_factor is the number of factors and n_epochs is the number of iteration of the SGD procedure. lr_all is the learning rate for all parameters and reg_all the regularization term for all parameters. In TABLE II, we can see the RMSE for every algorithm along with their optimal parameters.

Algorithm	Grid search
Baseline Only	bsl_options = sgd, als
Co-clustering	n_cltr_u : 1, 3, 5, 10; n_cltr_i = 1, 3, 5, 10
k-NN Basic	k = 20, 40, 70, 100, 150; min_k = 1, 3, 5; sim_options = user_based : true, false
k-NN Means	k = 20, 40, 70, 100; min_k = 1, 3, 5; sim_options = user_based : true, false
k-NN Baseline	k = 20, 40, 70, 100; min_k = 1, 3, 5; sim_options = user_based : true, false ; bsl_options = als, sgd
Slope One	no parameters
SVD	n_factor = 50, 100, 200, 300, 400; n_epochs = 50, 100, 200, 300; lr_all = 0.001, 0.004, 0.005, 0.01, 0.04, 0.05, 0.1; reg_all = 0.01, 0.04, 0.05, 0.1, 0.4, 0.5
NMF	n_factor = 50, 100, 200, 300; n_epochs = 50, 100, 200, 300

TABLE I: Range of every hyper-parameters for each algorithm during the grid search.

Algorithm	RMSE	Optimal parameters
Baseline only	1.00127	bsl_options = als
Co-clustering	1.00882	n_cltr_u = 1; n_cltr_i = 1
k-NN Basic	1.0.2282	k = 150; min_k = 3; user_based = true
k-NN Means	0.99948	k = 70; min_k = 5; user_based = false
k-NN Baseline	0.99936	k = 40; min_k = 5; user_based = false, bsl_options = als
Slope One	no calculation	no parameters
SVD	0.99614	n_factor = 400; n_epochs = 300; lr_all 0.004; reg_all = 0.1
NMF	1.00228	n_factor = 300; n_epochs = 200

TABLE II: RMSE score and optimal hyper-parameters of the grid search.

C. Data set splitting for the second stage

As we anticipated that we would perform a combination of the prediction algorithms, the dataset is split into two parts. Thus, 70% of the training set is used to train the algorithms of the first stage while the last 30% is used to determine the second stage. In this manner we avoid that information is shared between the test and training datasets, which can lead to over-fitting.

IV. Stage 2: Combination of algorithms

A. Linear Combination

For a given pair of user and item for which we want to estimate the prediction, we now have several possible values and we to decide how to give a final prediction. Instead of always taking the values given by the algorithm that reached the lowest RMSE we do a linear combination of the predictions given by all the algorithms hoping to compensate the individual error of each algorithm. Before applying a model we decide to remove any outliers from our previous prediction by forcing the lowest value to be 1 and the highest to be 5. We decide to use a ridge regression model and perform a cross validation with a k-fold of 10 on the linear space from 10^{-8} to 1 to find the best regularization parameter. The goal of our model is to find the weight vector w that will minimize the function :

$$||Xw - y||^2 + \lambda||w||^2$$

Where y is the rating, X represent the predictions of the different algorithms and λ is the regularization term that we also have to estimate. The training is done on the 30 remaining percent of the data-set with a standardized matrix X. This combination allow us to reach a RMSE of 1.028 on AIcrowd. This is a worse result than the RMSE of tableII but it is expected as it is only possible to send integer prediction between 1 and 5. The table III gives the parameters returned by the ridge regression. We can see that the weight attributed to SVD is higher than the other, which makes sense as it has

the highest a priori probability to be right (lowest RMSE in Table II). The regression parameter λ for this ridge regression is 1. Quite surprisingly the very simple and fast model Slope One as one of the highest weight whereas the slow k-NN Basic has the lowest. This is coherent with the fact the k-NN Basic has the highest individual RMSE in Table II and underline that k-NN Basic is not a very good model for our data.

Algorithm	Weights
Baseline only	-0.1126
k-NN Basic	0.0895
k-NN Means	-0.0233
k-NN Baseline	0.1403
SVD	0.3281
NMF	0.1386
Slope One	0.1460
Co-clustering	-0.1531

TABLE III: Weights returned by the ridge regression.

B. Features Expansion

As we have trained only 8 algorithms, we do not have a lot a of features to use. One solution to this problem would be to use more algorithms. However this is time consuming as we need to create (or use other libraries) and then train them to find the best parameters for each new algorithm. To overcome this problem we use polynomial expansion. This is done by adding to the matrix X the square, the cube and the exponent 4 of each features. The multiplication between each features if also added. By doing this we hope to highlight some new relevant features. The same cross validation as above is performed. We do not provide coefficient for each feature as it would be long and not be relevant.

V. Results

Running all the above algorithms and combining them with the ridge regression and a feature expansion we reach a RMSE of 1.026 on AICrowd. We realise that the feature expansion was a good idea as it makes us gain 0.002 in RMSE. The distribution of our predictions is different compared to the original one. Due to the quadrictic increase of the RMSE they are concentrated around the median of the initial dataset to avoid paying too much in case of an error. This is why we only have almost 0 (131) item rated to 1; if such a prediction is wrong the impact on the final score will be high.

VI. Discussion & Improvements

We can affirm that our results could have surely be improved if we had access to stronger computational power as a lot of algorithms were tested on small grids. This would have allowed us to also train other algorithms. The degree chosen for the expansion of the feature in the second layer has been set relatively arbitrary, it could be interesting to include the degree of the expansion in the cross validation. Another path that could have been explored would be that instead of always combining all the algorithms together, it could be interesting to combine only certain algorithms together and ignore others, in order to classify those which deteriorate the predictions and

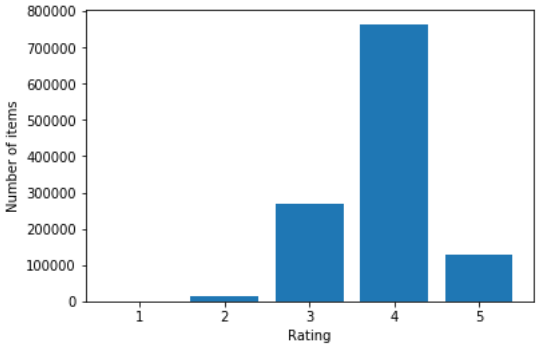

Fig. 3: Distribution of the five ratings in our predictions

those who improve the predictions. Finally, the implementation of a neural network model could definitively enhance the results.

VII. Conclusion

To conclude this project, among the many models based on Collaborative Filtering that we have implemented, we observed that SVD showed the best results in terms of individual performance. The predictions can be improved by combining these algorithms with few others, which approaches the solution that won the Netflix Prize. The best performance that we could achieve with our model gave a RMSE of 1.026 on AIcrowd .

VIII. Acknowledgements

We would like to thank Martin Jaggi, Rudiger Urbanke and all the teaching assistants for their careful reading and helpful suggestions.

References

[1] N. Hug, “Surprise, a python library for recommender systems,” http://surpriselib.com, 2017.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.