Recommender System: The Netflix Prize

Allan Bellahsene, Lionel Brodard, Antoine Marchal
CS-433 Machine Learning, EPF Lausanne, Switzerland

***Abstract*—Identifying the different tastes of consumers has always been a significant task for companies to increase their profits. Recommender systems are a way of using known consumers' tastes to predict unknown tastes. The Netflix Prize aims at predicting the ratings the users would give to movies they have not watched (or have not graded yet) based on ratings of movies they did watch. Several approaches can be used to solve this problem; a few of them are presented in this paper.**

I. Introduction

The whole purpose of a recommender system is to suggest to users items they *might* like. The higher the probability that the users like the recommended item, the better the recommendation system. Fortunately, we can have an idea on the power of prediction of our system even before it is in the hands of the users: simply by testing it on labels we already have, then calculating our error margin. In this project, independently on the model used, our accuracy metric will be the so-called Root Mean-Squared Error (RMSE).

All the models we will present fit in the collaborative filtering class of models, as we only have at our disposal past interactions between users and items, i.e., ratings. This class of models can be divided into two sub-categories: the memory-based and model-based approaches.

After quickly describing the data used for this challenge, a section will be dedicated to discussing the tuning of hyperparameters. While this task can often be left aside, we believe its dramatic influence on the performance of each method has to be highlighted. The relationship between hyperparameters and the RMSE is not always clear, and while grid searching for the best hyperparameter is often the path chosen, we will show that an alternative exists, much faster and surprisingly accurate: Bayesian Optimization. We will then dive in the core of our project by presenting the models we used that fit in the memory-based category, after quickly explaining the idea underlying this paradigm. Then, we will follow a similar procedure to present the models we used that fall in the second sub-category, the model-based method. Afterwards, we will present a model that combines all the models we used previously: we will use linear regression to find the optimal weights in which we should use each method. In this linear regression, each possible combination of features will be used: the combination that yields the lowest RMSE will be retained as our best combination. Finally, we will discuss the results obtained with each method.

It is important to note that throughout all this project, we used the 'Surprise' library [1] to perform many of our simulations, as this library has been designed for recommender system problems such as this one.

II. The Data

Our data consists of 10'000 users and 1'000 movies. We are only given 1'176'952 ratings, whereas we could theoretically have 10'000'000 ratings. Hence, we only have $\approx 11.77\%$ observed labels, and our goal is to use them to predict the remaining 88.23% ones. So we transpose our initial dataset into a matrix X of size $D \times N$, where D = 1000 and N = 10000, such that entry $x_{d,n}$ of the matrix corresponds to the rating given by user n for movie d. This matrix is, therefore, mostly a sparse matrix.

To split our data, we considered a 90-to-10 train/test ratio, following the procedure used in Lab. 10.

III. Hyperparameters

The choice of hyperparameters is often neglected, as it might not be the most exciting task in a machine learning project. Nevertheless, hyperparameters have a direct influence on the performance of a model, thus finding the hyperparameter that optimizes the loss function is very important. Figure 1 allows one to see that not only the RMSE varies quite significantly with k, but more importantly the relationship between k and the RMSE can differ even if the method used is the same (in this case, KNN), but only a parameter varies (in this case, the similarity coefficient).

However, an important thing to keep in mind is that Figure 1 represents the relationship between the RMSE and the hyperparameters only for a finite range of k, in this case between 40 and 60. This figure does not tell us what would the RMSE be for k outside this range. But it is something we would like to know, as it likely the hyperparameter k that minimizes the RMSE is outside this very restricted domain.

In fact, Bayesian Optimization [2] is one way to approach this problem. The goal is to pick random points in a given interval, that can be large enough to cover the entire domain, to approximate the true relationship between a variable and its function on the full domain of the function. As one can see on Figure 2, Bayesian Optimization allowed us to have an approximation of the relationship between k and the RMSE on the entire domain of k, allowing us to identify that $k_{min} \approx 309$, where k_{min} is at the very least a local minimum

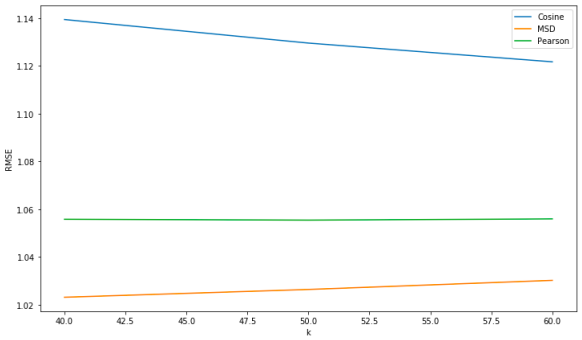

Fig. 1. Variation of the RMSE as a function of k using Grid Search, for different similarity measures

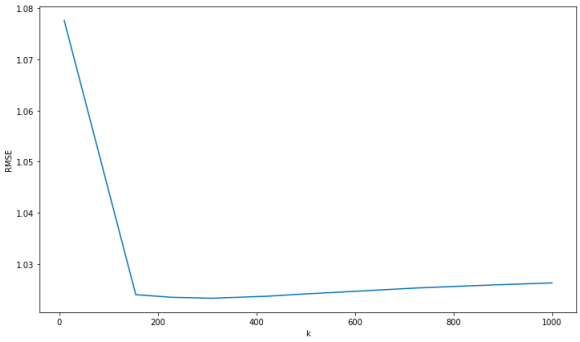

Fig. 2. Variation of the RMSE as a function of k using Bayesian Optimization

of the RMSE.

Hence, these results convinced us to use Bayesian Optimization when tuning the hyperparameters.

IV. Memory-based methods

Memory-based methods rely on the assumption that no underlying model explains the interactions between users and items. Based on observed interactions, this method simply relies on similarities between users and items to predict unobserved interactions. Naturally, the k-nearest neighbours algorithm appears as an evident algorithm that fits in this kind of models, and it is indeed the first model we used for our recommendation system. There are two ways to implement this algorithm: a user-user approach and an item-item approach. We tried both approaches.

A. K-nearest neighbours with a user-based approach

In the K-NN user-based approach, we focus on the similarities between users. Two users will be considered to be neighbours if they give similar ratings to most movies they both watched. The predicted rating given by user u for movie i is given by

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

where $sim(u,v)$ is the similarity between user u and user v, and r_{vi} is the observed rating given by user v for movie i. The subset $N_i^k(u)$ is the subset of movies watched by user u. The goal is hence to find users which subset of movies watched belong in this subset $N_i^k(u)$. The higher the movies in common shared by user u and user v, the more weight should be given to the similarity coefficient between the two users. What we mean is that if users A and B have a 100% similarity measure but they only watched one movie in common, this is less informative than if, say, users A and C have a 90% similarity measure but watched 30 movies in common. Obviously, user C should be considered as a nearer neighbour to user A, than user B. We denote by k the number of movies watched in common by a pair of two users. If we set $k = x$ to compute our predicted rating, the algorithm will look, for each pair of users, for x movies in common. If there are not x movies in common watched by a pair of users, the algorithm will look for $x - 1$ movies in common, and so on. As one can see in Figure 1, we decided to use the following similarity measures:

1) The cosine similarity: It measures the cosine of the angle between two vectors. The result will be bounded in $[0, 1]$ since we only have positive ratings.

$$sim_{cos}(i,j) = \frac{\sum_{u \in U_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{u \in U_{ij}} r_{ui}^2} \cdot \sqrt{\sum_{u \in U_{ij}} r_{uj}^2}},$$

for the Item-based approach, and

$$sim_{cos}(u,v) = \frac{\sum_{i \in I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I_{uv}} r_{ui}^2} \cdot \sqrt{\sum_{i \in I_{uv}} r_{vi}^2}},$$

for the User-based approach.

2) The Mean Squared Difference similarity: defined for both approach as

$$sim_{msd}(u,v) = \frac{1}{msd(u,v)+1}$$
$$sim_{msd}(i,j) = \frac{1}{msd(i,j)+1}$$

where,

$$msd(u,v) = \frac{1}{|I_{uv}|} \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2$$
$$msd(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} (r_{ui} - r_{uj})^2$$

3) The Pearson correlation coefficient: this is the classical Pearson correlation estimator:

$$sim_{corr}(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)(r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}$$

for the Item-based approach, and

$$sim_{corr}(i,j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)(r_{uj} - \mu_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)^2} \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \mu_j)^2}}$$

for the User-based approach. For each method, we tuned in k to minimize the RMSE. Using Bayesian Optimization, we obtain the following optimal hyperparameters:

- The cosine similarity: $k^* = 309$
- The Mean Squared Difference similarity: $k^* = 35$
- The Pearson correlation coefficient: $k^* = 35$

The results will be discussed in the adequate section.

B. K-nearest neighbours with an item-based approach

The item-based approach consists of finding, for each movie, its nearest neighbour. Two movies will be considered as neighbours if most of the users that rated them gave them similar ratings. So in this case, similarity is calculated across movies rather than users. The predicted rating given by user u for movie i is given by $\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} sim(i,j) r_{uj}}{\sum_{j \in N_u^k(i)} sim(i,j)}$,

where $sim(i, j)$ is the similarity between movie i and movie j.

Similarly, as in the user-based approach, the parameter k has a major role in the error margin and describes, in this case, the number of users that have watched the same movie. Again, the value of way k affects the RMSE depends on the similarity coefficient used. We also predicted the missing values of ratings using all types of similarity coefficients and, once again, obtained different results which will be discussed in the results section.

At this point, even before discussing the results, one can already questions on the suitability of K-NNs algorithms for recommender systems, as the ambiguity played by the hyperparameter k seems quite important, for both user and item-based approaches. And as we will see later on, it appears that model-based methods are indeed better suited for such problems.

Finally, we obtained the following optimal hyperparameters:

- The cosine similarity: $k^* = 309$
- The Mean Squared Difference similarity: $k^* = 309$
- The Pearson correlation coefficient: $k^* = 723$

V. Model-based methods

As opposed to memory-based methods, model-based methods rely on the assumption that the interactions between users and items can be explained by an underlying model. This underlying model directly implies that the sparse matrix of users-items interactions can be written as a product of two matrices, with lower dimensions. I.e., the interaction matrix $\mathbf{X}$ can be written as $\mathbf{X} = \mathbf{W}\mathbf{Z}^\mathbf{T}$, where $\mathbf{W}$ is a $D \times K$ matrix and $\mathbf{Z}$ is a $N \times K$ matrix. It is assumed that $K(<< N, D)$ is the number of features that can distinguish different movies and can also describe users' preferences. For example, if we assume $K = 2$, it can represent the two features:

- *Category of the movie is Y*
- *Actor F plays in the movie*

Then, the matrix $\mathbf{W}$ will 'tell' us whether each movie belongs in the category Y of movies or not, and whether *Actor F* plays in each movie or not. The matrix $\mathbf{Z}$ will tell us how much each user likes *category Y* of movies, and how much each user likes *Actor F*. Then, the product of these two matrices will be used to predict the ratings of the users for movies they have not watched such that:

$$X \approx W \cdot Z^T = \hat{X}$$

Therefore, the prediction for the d^{th} movie for the n^{th} user is given by:

$$\hat{x}_{dn} = w_d^T \cdot z_n = \sum_{k=1}^{K} w_{dk} \cdot z_{nk}$$

We then round the prediction to get an integer between 1 and 5. For values below 1, the predicted value is rounded up to 1. For values above 5, the predicted value is rounded down to 5. We used both the Stochastic Gradient Descent (SGD) and the Alternating Least Squares (ALS) methods when performing the matrix factorization.

A. Matrix Factorization with Stochastic Gradient Descent

As seen in the lecture 10a and lab 10 of the *Machine Learning CS-433* EPFL course, we can use the Stochastic Gradient Descent to factorize the matrix. The utility of using the Stochastic version is that it has the property of being local, which reduced the cost of computation. The cost function to minimize is given by:

$$\begin{aligned} L(W, Z) =& \frac{1}{2} \sum_{\Omega \in (d,n)} (x_{dn} - (WZ^T)_{dn})^2 \frac{\lambda_w}{2} ||W||^2_{Frob} \\ & - + \frac{\lambda_z}{2} ||Z||^2_{Frob} \end{aligned}$$

where λ_w and λ_z are parameters for the regularization term for the movies matrix and the user matrix respectively. We update both W and Z according to:

$$\begin{aligned} e_{dn}^2 &= (x_{dn} - \hat{x}_{dn})^2 \\ w'_{dk} &= w_{dk} + \gamma(e_{dn} z_{nk} - \lambda_w w_{dk}) \\ z'_{nk} &= z_{nk} + \gamma(e_{dn} w_{dk} - \lambda_z z_{nk}) \end{aligned}$$

The hyperparameters we used are the following ones:

$$\begin{array}{lll} \gamma = 0.008 & \lambda_w = 0.01 & epochs = 0.01 \\ K = 20 & \lambda_z = 0.01 & \end{array}$$

B. Matrix Factorization with Alternative Least Squares

We base our ALS implementation on the laboratory session 10 of the *Machine Learning CS-433* EPFL course. We randomly fill W and Z. Then, we minimize W with respect to (w.r.t) Z before minimizing Z w.r.t W. The process is repeated until the loss function reaches a certain threshold. The loss function is the same as for the SGD version, which is:

$$\begin{aligned} L(W, Z) =& \frac{1}{2} \sum_{\Omega \in (d,n)} (x_{dn} - (WZ^T)_{dn})^2 \frac{\lambda_w}{2} ||W||^2_{Frob} \\ & - + \frac{\lambda_z}{2} ||Z||^2_{Frob} \end{aligned}$$

We update both W and Z according to:

$$W = (W^T W + \lambda_w I_k)^{-1} W^T X$$

$$Z = (Z^T Z + \lambda_z I_k)^{-1} Z^T X$$

The hyperparameters used are:

$$\lambda_w = 0.085 \qquad \lambda_z = 0.085$$
$$K = 20 \qquad \text{stop-criterion} = 10^{-5}$$

VI. Ensemble

The ensemble method consists in implementing the following linear regression:

$$\mathbf{Y^{set}} = \alpha + \sum_{i=1}^{k} \beta_i \hat{Y}_i^{set} \tag{1}$$

where $\mathbf{Y^{set}}$ is the vector of observed ratings, the index $set = \{test, train\}$ denotes whether the observed ratings come from the test or the train set, $\hat{Y}_{i,set}$ corresponds to ratings predicted by model i given the dataset *set*, β_i corresponds to the optimal coefficient in which model i is used to predict the true ratings, and α is a vector of ones to introduce a constant in our linear regression.

After splitting our data in a 90-to-10 ratio, we ran the following regression:

$$\begin{aligned}\mathbf{\hat{Y}^{train}_{ensemble}} =& \alpha + \beta_1 \hat{Y}^{train}_{MFALS} + \beta_2 \hat{Y}^{train}_{MFSGD} + \\ & \beta_3 \hat{Y}^{train}_{KNNIBC} + \beta_4 \hat{Y}^{train}_{KNNIBP} + \beta_5 \hat{Y}^{train}_{KNNIBM} + \\ & \beta_6 \hat{Y}^{train}_{KNNUBC} + \beta_7 \hat{Y}^{train}_{KNNUBP} + \beta_8 \hat{Y}^{train}_{KNNUBM}\end{aligned} \tag{2}$$

where each subscript under $\hat{Y}^{train}$ is the name of the method used (e.g., *KNNIBC* means *KNN Item Based Cosine*).

After performing this regression on our train data, we obtained our estimated matrix $\hat{\beta}$. We then use this matrix to compute $\mathbf{\hat{Y}^{test}_{ensemble}}$, and we then calculate the RMSE by computing the difference between this quantity and the observed $\mathbf{Y^{test}}$, and using a ridge regression in order for our model not to overfit.

The idea we wanted to implement at this point of the project was to perform the regression on every possible combination of methods we implemented before. This means equation (2) is in fact one combination possible out of the $\sum_{i=1}^{k} C_i^k$ possibles, where $k = 8$ is the number of models, and $C_i^k = \frac{k!}{i!(k-i)!}$. In this case, we thus tested 255 possible combinations and retained the one that minimizes the RMSE. Results will be discussed in the following section.

VII. Results

During this project, we tried to implement a large variety of methods in order to predict in the best way possible what the users of an interactive platform such as Netflix would enjoy. To do so, we approached the problem using two main paradigms: memory-based methods and model-based methods.

Method	Submission ID	RMSE
KNN - UB - C	30242	1.064
KNN - UB - M	30245	1.055
KNN - UB - P	30246	1.064
KNN - IB - C	30227	1.116
KNN - IB - M	30229	1.058
KNN - IB - P	30231	1.086
MF - ALS	30215	1.025
MF - SGD	30234	1.040
Blended	30219	1.036

TABLE I
RMSEs obtained for the different methods used

For the memory-based methods, we tried to implement the famous K-Nearest neighbour's algorithm. We presented a few variants of this algorithm, depending on whether we should predict the neighbours based on items or users. For both these cases, there were also variants among them, as the similarity coefficient was calculated using Cosine Similarity, Mean Squared Difference, or Pearson Similarity. We thus implemented six different methods in the range of memory-based methods. The best RMSE provided by each of this method can be read on Table I. As discussed in section III, the choice of hyperparameters had a significant influence on each of these results, and even though we tried to optimize their choice using Bayesian Optimization, this could only provide us with an approximation of the relationship between the hyperparameters and the RMSE of each method. There is, of course, a chance we did not find the hyperparameters that were truly minimizing the RMSE. In an ideal world, we would have used Grid Search and tried every point possible, but of course, this is unfeasible with our limited resources. But in the end, as one can see on the table above, the results obtained with memory-based methods were quite disappointing, as our best method, the KNN User-Based MSD method, only yielded an RMSE of 1.055.

Using Matrix Factorization, we managed to obtain way more promising results. Alternative Least Squares Matrix Factorization indeed allowed us to obtain a decent RMSE of 1.025, which allowed us to be ranked 24th out of 42 on the *AICrowd* platform (under the nickname *QatariMen*, the 18th December at 12:19).

The blended (or ensemble) method, discussed in section VI, resulted in a quite disappointing RMSE as we could have imagined it to be our best method for its 'ensemble' characteristics. To implement it, we tried 255 combinations as discussed in the section, but we could not get a lower RMSE than the one obtained by solely using ALS MF.

Finally, if we were to continue working on this project, applying neural networks would naturally be our priority. Particularly, using neural networks embeddings would be very interesting in such a project. Indeed, according to the

specialized website TowardsDataScience.com, *"embeddings are a way to represent discrete - categorical - variables as continuous vectors. (...) they place similar entities closer to one another in the embedding space."* Hence, this could be a way to improve our KNN algorithms.

REFERENCES

[1] Nicolas Hug. *Surprise, a Python library for recommender systems*. http://surpriselib.com. 2017.

[2] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*. 2012, pp. 2951–2959.