CS433-Machine Learning Project 2 : Recommender System

Gerald Sula - Ridha Chahed - Walid Ben Naceur
Department of Computer Science, EPFL, Switzerland

***Abstract*—In this project, we study the problem of building an efficient recommender system. We are only given access to a subset of users and their ratings, and we aim to recommend new movies by predicting the missing ratings. To this end, we considered a collaborative-based filtering approach along with ensemble methods. We trained 15 models, with SVD outperforming the other ones. By using feature augmentation and blending techniques such as XGBoost in order to build a robust predictor, we achieved a score of 1.025 on AI-crowd's validation set.**

I. Introduction

A recommender system seeks to predict the rating or preference a user would give to an item. The aim of this project is to build a recommender system where the users are Netflix customers, and items are movies proposed by Netflix. Recommender systems often rely on **collaborative filtering**, which relies only on past user behaviour- for example, users' previous product ratings- and does not require the creation of explicit profiles. In order to establish recommendations, collaborative filtering systems need to compare two different types of objects : items against users. There are two main approaches to carry out those comparisons :

- **The neighborhood approach** : this method is centered on computing the relationships between items, or alternatively, between users. An item-item approach evaluates the preference of a user for an item based on ratings of similar items based on ratings of similar items by the same user. This method transform users to the item space by viewing them as baskets of rated items. This way, we no longer need to compare users to items, but rather directly relate items to items.
- **Latent factor models** : Those methods, such as Matrix Factorization or Singular Value Decomposition, consist in an alternative approach by transforming both items and users to the same latent factor space, thus making them directly comparable. The latent space tries to explain ratings by characterizing both items and users on factors automatically inferred from user feedback. For example here, factors might measure obvious dimensions such as the amount of comedy, action, orientation to children, or duration of a movie, less well defined dimensions such as depth of character development, or completely uninterpretable dimensions. For users, each factor measures how much the user likes movies that score high on the corresponding movie factor.

We have used methods from exploratory data analysis, feature processing, hyper-parameter estimation through cross validation, visualization, implementation of collaborative filtering based algorithms and ensemble learning to obtain the results that we will describe in the following sections of this paper. Overall, we have used 15 models to predict new ratings on movies by users, and we have tried 4 different ways to blend the models, before choosing XGBoost in a similar way as BellKor's Pragmatic Chaos[2],the winners of the 2009 Kaggle competition from which this challenge is taken from. 10 out of the 15 models were directly implemented in the Surprise Python library, and the remaining 5 were implemented by ourselves.

II. Exploratory data analysis

A. Data description

The data-set is composed of a list of indices (user; movie) and the rating of that movie from that user, an integer between 1 and 5. There are N = 10000 users and D = 1000 movies, but just 1,176,952 evaluations are available (approximately 11.8%). Our goal is to find a model that can accurately predict the missing evaluations. We extensively explored the data, but we will only discuss some of our findings. All this information is condensed in table 1.

Table 1 : Statistics of the Netflix data-set

Number of users	10,000
Number of movies	1,000
Number of given ratings	1,176,952
Average of all ratings	3.857
Sparsity	88.23 %
Median of ratings per user	104
Median of ratings per movie	880

B. User behaviour

We wondered if the behaviour of users was biased, if there were inactive users that did not rate any movie, or if there was any particular rating pattern given by users, i.e multiple voters that only give 1 star ratings. If a high amount of those users were to be present in the data-set, this could bias the learning algorithms. Figure 1 shows a histogram of the distribution of users that have rated movies, and confirm users' good participation. Figure 2 shows the non-uniform distribution of the ratings given by all users. We can see that 35,743 1-star ratings are given to movies, whereas 400,852 5-star ratings were given. This plot shows that when users give ratings to movies, they usually give a high rating. We wondered also if the average grade of movies was skewed, and if some transforms had to be applied on the data. Figure 3 shows that the movies' averages are normally distributed, with an average rating of 3.5 and a standard deviation of 0.5.

Figure 1 : Distribution of users that have rated movies

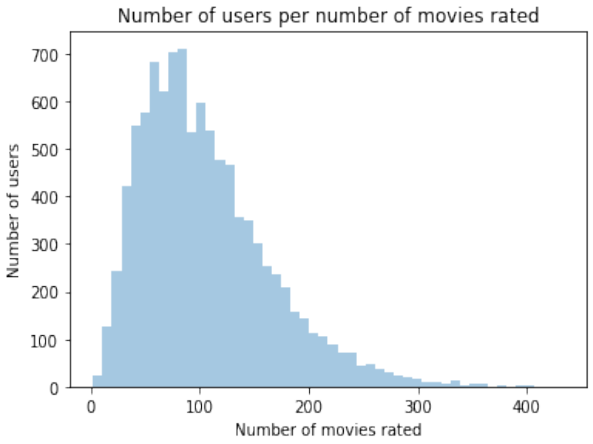


Figure 2 : Distribution of movies per per number of rating received

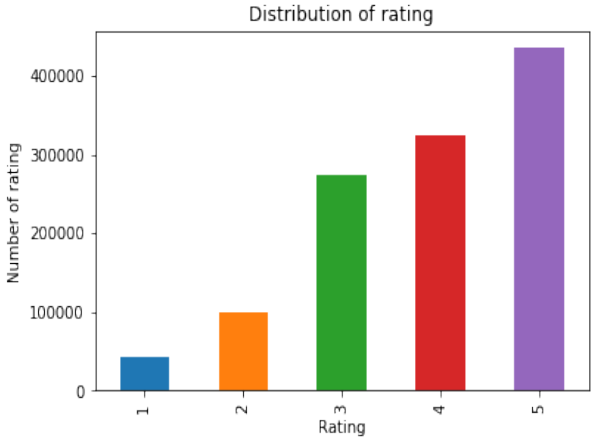
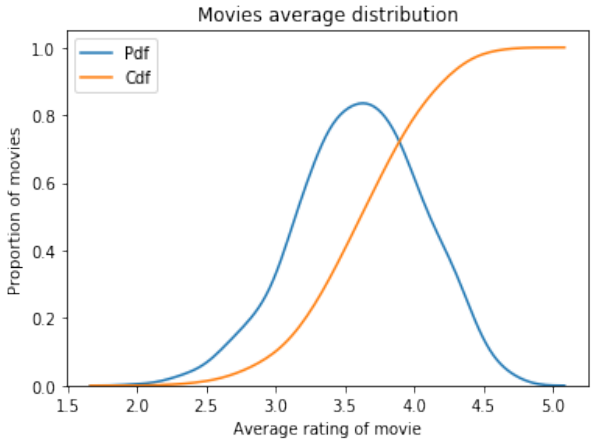


Figure 3 : Distribution of the movies' averages



III. Models and Methods

We present in this section all the models we have trained, prior to the blending of the models. We will describe verbosely SVD and ALS, since SVD was the model that performed best and ALS was implemented by our team following the guidelines of homework 10. More complete information on the other models can be found in [1].

A. Global Mean

A simple model is to compute the mean value of all the non-zero ratings in the training set, and return this value as a prediction.

B. User and Movie Mean

A better approach than the global mean model is to consider the average rating given by each user for all the movies they have rated and predict that score for the movies that this user has yet to rate. Similarly, we do the same for the movies : for each movie, we compute the average rating given by all users that have rated that movie, and predict that value as a rating for any user that has yet to rate that movie. The three means are at the same time considered as models, and as extra features in our feature expansion scheme described in section IV.

C. Baseline

Typical collaborative filtering data exhibit large user and item effects, i.e systematic tendencies for users to give higher ratings than others, and for some items to receive higher ratings than others. To cope with this issue, we need to adjust the data, and this is done by using **baseline estimates**. Let μ be the overall rating. A baseline estimate for an unknown rating $r_{u,i}$ is denoted by $b_{u,i}$, that we compute the following way :

$$b_{u,i} = \mu + b_u + b_i$$

Here, b_u and b_i indicate the observed deviations of user u and item i from the average μ. Learning those parameters is done by solving a least square problem by gradient descend. This algorithm is implemented in the Python Surprise library[1].

D. Co-Clustering

The Co-Clustering collaborative filtering algorithm assigns to users and items some clusters C_u , C_i, and $C_{u,i}$.

Clusters are assigned using a straightforward optimization method, much like k-means, where the number of clusters is a hyper-parameter to optimize.

This algorithm is implemented in the Python Surprise library[1].

E. SVD (Matrix Factorization) and SVD++ (Matrix Factorization with implicit ratings)

SVD is a matrix factorization with SGD, and it includes the learning of biases in the least square problem. This allows to take into account latent unknown features that are not accounted by the matrix features. Conceptually Singular Value Decomposition (SVD) decomposes the matrix X into W and Z of the given rank which minimizes the sum-squared distance to the target matrix X.

The prediction $\hat{r}_{u,i}$ is set as:

$$\hat{r}_{u,i} = \mu + b_u + b_i + q_i^T p_u$$

b_u, b_i, q_i^T, p_u denote respectively the user factors, the item factors, the user biases, and the item biases (here and in the following algorithms as well).

If user u is unknown, then the bias b_u and the factors p_u are assumed to be zero. The same applies for item i with b_i and q_i. To learn all the unknown, we minimize the following regularized squared error:

$$\sum_{r_{u,i} \in R_{train}} (r_{u,i} - \hat{r}_{u,i})^2 + \lambda(b_u^2 + b_i^2 + \|q_i\|^2 + \|p_u\|^2)$$

The minimization is performed by stochastic gradient descent. Let $e_{u,i} = r_{u,i} - \hat{r}_{u,i}$

$$b_u \leftarrow b_u + \gamma(e_{u,i} - \lambda b_u)$$
$$b_i \leftarrow b_i + \gamma(e_{u,i} - \lambda b_i)$$
$$p_u \leftarrow p_u + \gamma(e_{u,i} \cdot q_i - \lambda p_u)$$
$$q_i \leftarrow q_i + \gamma(e_{u,i} \cdot p_u - \lambda q_i)$$

Baselines are initialized to 0. User and item factors are randomly initialized according to a normal distribution. γ and λ are hyper-parameters determined using cross-validation. This algorithm is implemented in the Python Surprise library.

The SVD++ algorithm, an extension of SVD taking into account implicit ratings. Here, an implicit rating describes the fact that a user u rated an item j, regardless of the rating value. The prediction $\hat{r}_{u,i}$ is given by :

$$\hat{r}_{u,i} = \mu + b_u + b_i + q_i^T(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j)$$

This method is also implemented in the Python Surprise library[1].

F. NMF : Non-negative Matrix Factorization

A collaborative filtering algorithm based on Non-negative Matrix Factorization. This algorithm is very similar to SVD. The prediction $\hat{r}^{u,i}$ is set as :

$$\hat{r}_{u,i} = q_i^T p_u$$

where user and item factors are kept positive. The optimization procedure is a (regularized) stochastic gradient descent with a specific choice of step size that ensures non-negativity of factors, provided that their initial values are also positive.

The whole implementation of this algorithm is from the Python Surprise library[1].

G. KNN : item/user based

These K-nearest neighbors models build similarity matrices respectively over the movies/users and utilize them to find the k most similar movies/users to a given one. The distance metric used can be either cosine, mean squared difference or pearson. We have used the pearson baseline similarity metric for this project.

A weighted average of neighbors are then used to predict the elements according to their similarities. Those algorithms are implemented in the Python Surprise library [1].

H. Matrix Factorization - SGD and ALS

Given D items, N users and the corresponding rating matrix $X \in R^{D\times N}$, matrix factorization model aims to decompose the rating matrix into two lower rank matrices $W \in R^{D\times K}$ and $Z \in R^{K\times N}$.

For **Stochastic Gradient Descent** (SGD),the training objective is a sum over $|\Omega|$ terms. We want to minimize the function $L(w, z)$, where :

$$L(w,z) = \frac{1}{|\Omega|} \sum_{(d,n)\in\Omega} \frac{1}{2}[x_{d,n} - (WZ^T)_{d,n}]^2$$

Let $f_{d,n} = \frac{1}{2}[x_{d,n} - (WZ^T)_{d,n}]^2$
SGD solves the minimization problem by converging almost surely to a local minimum by doing the following updates at each step t :

$$W^{t+1} = W^t - \gamma \nabla_W f_{d,n}(w,z)$$
$$Z^{t+1} = Z^t - \gamma \nabla_Z f_{d,n}(w,z)$$

Alternative Least Squares (ALS) is an alternative to the SGD to the matrix factorization problem. We try here to minimize the following quantity :

$$\frac{1}{2} \sum_{(d,n)\in\Omega} [x_{n,d} - (Z^T W)_{n,d}]^2 + \frac{\lambda_w}{2}\|W\|^2 + \frac{\lambda_z}{2}\|Z\|^2$$

ALS is a two-step iterative algorithm. In every iteration, it first fixes Z and solves for W , and then it fixes W and solves for Z. Fixing one matrix at a time, the problem is reduced to a linear regression and a simple least squares technique can be used.

Those two algorithms were implemented following the indications in lab 10[3].

I. Slope One

The Slope One collaborative filtering algorithm takes into account both information from other users who rated the same item and from the other items rated by the same user. This algorithm is implemented in the Python Surprise library[1].

IV. Ensemble Learning : Blending the models

Our approach to this project was to use **ensemble learning**. Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. We split the data into two sets : a **model train set** and a **blending train set**, with 90% of the data for the former and the remaining 10% for the latter. The training for this part was done with those 10%. We implemented 4 ensemble methods, that we compare in this section. Results for each ensemble methods are describe in table 2.

Once we had the predictions from the 13 methods described in section III, we first decided to treat the blending problem as a **regression** task, with features being the predictions of each model. We tried to apply the **Least Squares** algorithm seen in the course.

We then decided to treat the blending problem as a multi-class **classification** task, where classes where the ratings in {1,2,3,4,5}. We used the **logistic regression** algorithms seen in class with L_2 regularization to avoid overfitting.

The ensemble method that gave the smallest RMSE is XGBoost The base learners of XGBoost are tree ensembles. The tree ensemble model is a set of **classification and regression trees (CART)**. Trees are grown one after another and attempt to reduce the misclassification rate made in subsequent iterations.

Minimize We also tried to find the weights of the weighted sum of the different models predictions by using the optimization method **Sequential quadratic programming (SQP)**, an iterative method for constrained nonlinear optimization.

Table 2 : RMSE of the different ensembling methods tried

Ensembling	**RMSE**
Least Squares	1.027
Logistic Regression	1.234
Minimize	1.028
XGBoost	1.025

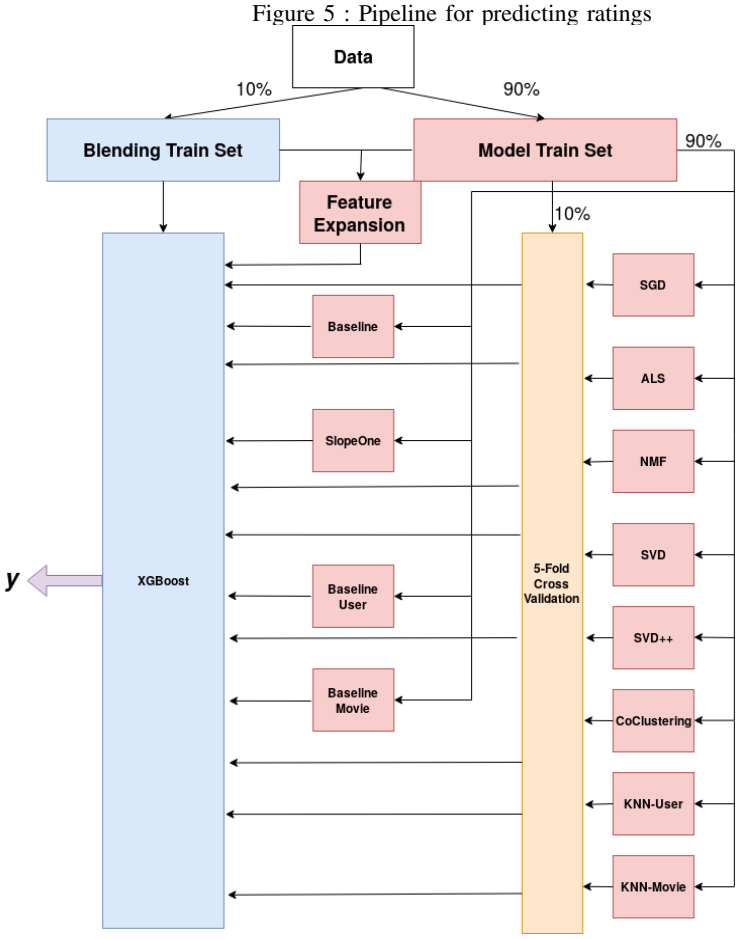
Feature Expansion : We have added the following 13 features in order to improve the ensembling method.

- Global Average : Average rating of all the ratings
- User Average : User's Average rating
- Movie Average : Average rating of this movie
- Similar users rating of this movie (cosine similarity) : SimUser1, SimUser2, SimUser3, SimUser4, SimUser5 (top 5 similar users who rated that movie..). For each similar user need to find the rating that he put for that movie if not available put the average rating of that user as an estimator.
- Similar movies rated by this user (cosine similarity): SimMovie1, SimMovie2, SimMovie3, SimMovie4, SimMovie5 (top 5 similar movies rated by this user..) For each similar movie we need to find the rating that the user has given to it if not available give the similar movie average rating.

V. Pipeline process

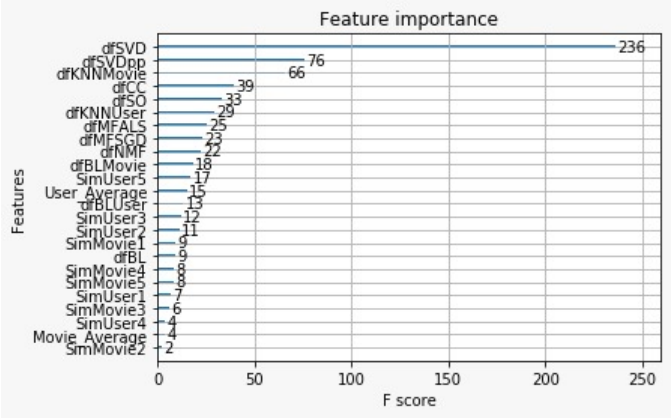
All of the above sections are summarized in the diagram on figure 5, detailing the split of the data for training the models, the cross-validation and the final blending. **y** denotes the final prediction that our model outputs.

Figure 5 : Pipeline for predicting ratings



VI. Results

Figure 6 : Importance of each model in the blending



The quality of the results for the Netflix-Recommender System challenge is measured by the Root Mean Squared Error (RMSE), and by a measure called **secondary**. RMSE is defined by :

$$\sqrt{\sum_{(u,i)\in TestSet} \frac{(r_{u,i} - \hat{r}_{u,i})^2}{|TestSet|}}$$

We mostly focused on improving the RMSE. Hyper parameter optimization was done following a 5-fold cross validation. All our results are summarized in the following table.

Table 3 : Results for our predictions

Model	**Local RMSE**	**RMSE**	**Hyper Parameters**
SlopeOne	0.992	1.060	N/A
CoClustering	0.990	1.067	n_cltr_u=13, n_cltr_i=13,n_epochs=200
SVD	0.994	1.027	n_factors=20, n_epochs=20, lr_all=0.002, reg_bu=0.1, reg_bi=0.01
SVD++	1.018	1.059	n_factors=20, n_epochs=20, lr_all=0.002, reg_bu=0.1, reg_bi=0.01
NMF	1.014	1.055	n_factors= 15,n_epochs= 50, reg_pu=0.06,reg_qi= 0.06, reg_bu= 0.02,reg_bi=0.06
KNN-Movie	1.060	1.078	k_movie = 300
KNN-User	1.019	1.074	k_user = 100
MF-ALS	0.994	1.028	num_features = 20, (K in the lecture notes) λ_user = 0.080, λ_item = 0.080, stop_criterion = 1e-5
MF-SGD	1.001	1.043	γ = 0.025 ,num_features = 20, (K in the lecture notes) λ_user = 0.1, λ_item = 0.01, num_epochs = 20
Baseline	1.014	1.041	N/A
Global Mean	1.122	-	N/A
Movie Mean	0.976	-	N/A
User Mean	1.031	-	N/A
XGBoost	-	**1.025**	N/A

Figure 6 shows the importance of each model in the overall XGBoost ensembling. The metric to evaluate the importance of features (in this case, our features are the predictions from the 13 models) is the F-Score.

VII. Summary and Discussion

We have shown that combining different methods can improve RMSE of the final prediction. While individual models score an RMSE of at least 1.027, blending achieves an RMSE of around 1.025. The blending improves the score since models can compensate for each other. Selecting an appropriate ensembling method is important : multiple techniques have been explored in this project, with XGBoost outperforming the others. Further improvements could be possible by adding more models for the blending process, considering different ensembling methods, better hyper-parameter optimization and expanding the feature space beyond what was already done, but this might add too much complexity in the overall model, which would be prone to overfitting.

References

[1] Surprise Python library : https://surprise.readthedocs.io/en/stable/prediction_algorithms_package.html, accessed : 2019-12-10

[2] Y. Koren, "The BellKor Solution to the Netflix Grand Prize," 2009.

[3] "CS433 epfl, machine learning course," https://mlo.epfl.ch/page-146520-en-html/, accessed: 2019-12-10.