Machine Learning Project II : Recommender System

Vincent Micheli, Ivan Schonenberger, Lionel Constantin
EPFL, Switzerland

***Abstract*—Implementation of a Collaborative Filtering Recommender System for movie ratings prediction. We propose an ensemble method using a blend of 10 algorithms inspired by Matrix Factorization and K-NN techniques. We achieved a RMSE of 1.018 on crowdAI's EPFL ML Recommender System challenge.**

I. Introduction

Recommender systems are one of the most successful and widespread application of machine learning technologies in the industry. In this paper they are used to recommend movies to users based on their tastes. We only have access to previous ratings and therefore resort to collaborative filtering algorithms. We first look for valuable insights by performing an exploratory data analysis. Then we propose a machine learning pipeline and different models to improve upon basic solutions. Ultimately each model is evaluated and we discuss the results obtained.

II. Data description and exploratory data analysis

A. Data description

The data is available on the EPFL ML recommender system challenge on crowdAI [1]. It is composed of a labeled training set (data_train.csv) and an unlabeled testing set (sampleSubmission.csv). Each line contains an user ID, movie ID and the associated rating. The labeled data contains a rating matrix from 10000 users on 1000 movies. Ratings range from 1 to 5 on an integer scale. In total we are given 1 176 952 ratings to train our models and the same amount to predict for the challenge. The ratings matrix is sparse with 88.23% of entries missing.

B. Exploratory data analysis

We first observe the distribution of the number of ratings given per user and received per movie.

Most ratings given	522
Least ratings given	3
Average ratings given	118
Median ratings given	104.0

Table I: Ratings given statistics

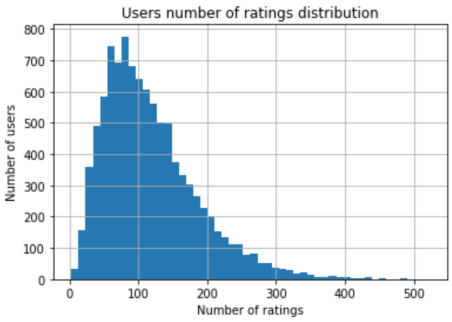

Figure 1: User number of ratings

Most ratings received	4590
Least ratings received	8
Average ratings received	1177
Median ratings received	883.5

Table II: Ratings received statistics

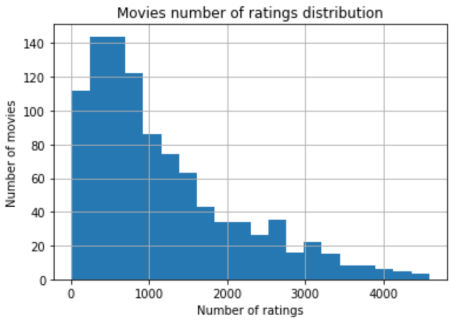

Figure 2: Movie number of ratings

Both histograms reveal right skewed distributions as suggested by a median to mean ratio smaller than 1 (to a greater extent for movies than for users). That is some users give and some movies receive an inordinate amount of ratings compared to their peers.
We then visualize the distribution of mean ratings for users and movies.

Highest user average rating	4.93
Smallest user average rating	1.85
Average user average rating	3.83
Median user average rating	3.85

Table III: User average rating statistics

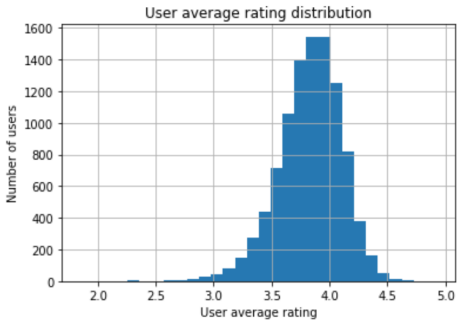

Figure 3: User average rating

Highest movie average rating	4.73
Smallest movie average rating	2.02
Average movie average rating	3.60
Median movie average rating	3.62

Table IV: Movie average rating statistics

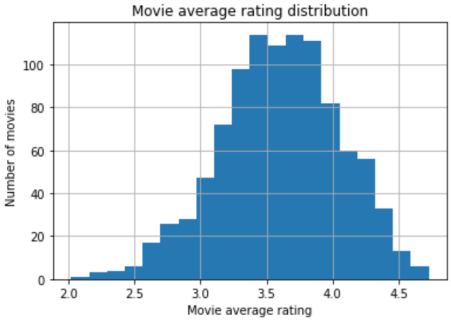

Figure 4: Movie average rating

Both histograms reveal user and movie bias. That is some users tend to give and some movies tend to receive higher ratings than others. We also note the absence of malevolent users, that is users that would give ones or fives to every movie they rated. Here we do not have to deal with this kind of corrupted data.

III. Models and Methods

In this section we describe our machine learning system. We first make general remarks about notation and implementation. We then describe the models considered, the optimization procedure used to train them and the determination of hyperparameters by grid search. Then we explain how we split our dataset into training and validation sets and our tuning procedure. Finally we summarize the grids searched to tune the hyperparameters for each model.

A. Notation and Implementation

We use the notation u for the users and i for the movies. r_{ui} is the rating of the u^{th} user on the i^{th} movie, and we define $\hat{r}_{ui}$ as the predicted rating. R is the sparse matrix of the data with coefficients r_{ui}.
To implement the ideas outlined in this paper we make extensive use of the Python Surprise Library [2] where most models are implemented and documented. If we do not explicitly mention the value of certain hyperparameters and do not mention tuning them then this means we use the library's default values which can be found in its documentation [2]. The reason we do so is that we cannot tune all possible hyperparameters as we are constrained in both time and computational power. Thus we concentrate our efforts on the hyperparameters we think will have a greater impact on our performance while resorting to the library's standards for the rest.

B. Global mean

The simplest solution is to compute the mean of all ratings contained in the training set, then output these values as predictions.

$$\hat{r}_{ui} = \mu = \frac{1}{|R|} \sum_{r_{ui} \in R} r_{ui}$$

We use this model as a benchmark to compare all subsequent models with.

C. User/Movie mean

Another approach is to compute the mean of the ratings by users or movies and output this as a prediction:

$$\hat{r}_{ui} = \mu_u = \frac{1}{|R_u|} \sum_{r_{ui} \in R_u} r_{ui}$$

where R_u is the set of all movies ratings for user u. Similarly we can compute $\hat{\mu}_i$, the average for a certain movie. In this case:

$$\hat{r}_{ui} = \mu_i = \frac{1}{|R_i|} \sum_{r_{ui} \in R_i} r_{ui}$$

where R_i is the set of all movie ratings for the i^{th} movie.

D. Baseline

The previous models are combined into the baseline model. That is predictions are made from the sum of the global mean and two parameters to estimate, the user bias and the movie bias. We define the baseline as such:

$$b_{ui} = \mu + b_u + b_i$$

The regularized loss function to be minimized can then be written as follows:

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

As our optimization procedure, we use SGD (stochastic gradient descent) and keep all default parameters except λ which we tune.

E. K-NN User/Movie based with baseline

For both models, each pair of users or movies is assigned a similarity metric $\mathrm{sim}()$. Then the baseline recommendation is complemented by the closest neighbourhood of the user or movie. For user based K-NN, we can write the model as follows:

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

where $N_i^k(u)$ is the set of the up to k closest neighbors of user u that have rated the i^{th} movie and for movie based K-NN:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{j \in N_u^k(i)} \mathrm{sim}(i,j)(r_{uj} - b_{uj})}{\sum\limits_{j \in N_u^k(i)} \mathrm{sim}(i,j)}$$

with an analog definition for $N_u^k(j)$.
In both cases, we use ALS (Alternating Least Squares) as our optimization procedure while simultaneously increasing the number of epochs to 50 for a better accuracy. We only tune the maximum number of neighbors considered k. The remaining hyperparameters are kept at default including the similarity measure which is the shrunk Pearson correlation.

F. Slope One

Next we consider a very simple model, which has no parameter to estimate and no hyperparameters. The algorithm is described in a paper by Lemire and Maclachlan [3]. The model predicts as follows:

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} \sum_{v \in U_{ij}} \frac{1}{|U_{ij}|}(r_{vi} - r_{vj})$$

where $R_i(u)$ is the set of items j rated by u that have a least one common user with i and U_{ij} is the set of all users that have rated both item i and j.

G. SVD without baseline

We now consider the classic matrix factorization model we have seen in class. The loss function is then:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - q_i^T p_u) + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

The hyperparameters we tune are the number of factors k and the rigde parameter λ. We use SGD as our optimization procedure and increase our number of epochs to 500 while decreasing the learning rate to 0.0015. We do so since this algorithm is not that computationally expensive and thus we can increase the accuracy of our optimization routine.

H. SVD with baseline

We take the same model as before but we now add a baseline to it. The loss function becomes:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - (\mu + b_i + b_u + q_i^T p_u)) + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$$

Our treatment of hyperparameters for this model is exactly the same as the version without baseline.

I. SVD++

This algorithm is an extension of SVD and was proposed by the BellKor team [4]. Its goal is to capture implicit ratings. In our case this may seem odd since we don't have access to metadata about the users or movies. Indeed the modification made to SVD is to represent the latent factors vector for each user as an original vector influenced by the movies rated. The prediction is now:

$$\hat{r}_{ui} = u + b_u + b_i + q_i^T(p_u + |N(u)|^{-1/2} \sum_{j \in N(u)} y_j)$$

where we take into account the implicit feedback present in the set of rated items $N(u)$.
This is by far the most computationally costly algorithm described in this paper. Therefore we keep all hyperparameters at default values as we do not have the resources to tune them.

J. Model Blending

We model the true rating prediction as a penalized linear combination of the predictions from the previously described algorithms. That is we perform a ridge regression based on individual predictions as covariates. The only hyperparameter is the ridge regularizer λ, which we tune.
Formally, each input vector x_i is the F-dimensional vector of the predictions in the ensemble where F is the number of models. For N data samples, one collects the vectors $x_1, ..., x_N$ in an $N \times F$ matrix X of predictions. The target values for these n data points are collected in an N-dimensional vector y. For any input vector x, the prediction is $\hat{y} = x^T w$ with weight vector $w = (X^T X + \lambda I)^{-1} X^T y$ [5].

K. Hyperparameters tuning and model evaluation

We split the training data into 2 distinct parts each containing respectively 80% and 20% of the ratings. We call the former the model tuning set and the latter the blender tuning set.
The model tuning set is used to train and tune each individual model. To find the best hyperparameters we use 3-fold cross validation and pick the tuple of hyperparameters in the grid search that yields the lowest average RMSE. Once we have found optimal hyperparameters for each individual model we retrain them on the whole model tuning set.

We move on to the blender tuning set. First we evaluate the performance of our models on this set. Therefore it also acts as validation set. Then we split this set into 2 distinct parts each containing respectively 15% and 5% of the ratings. They act as a model blending training set and a model blending validation set. To find the best λ regularizer we use 3-fold cross validation on the model blending training set and pick the constant yielding the lowest average RMSE. Finally we evaluate the performance of our ensemble method on the model blending validation set.

We are left with the challenge predictions. We retrain each individual model on the whole training data with the previously picked hyperparameters. Then for each algorithm we predict ratings for every pair of user/movie in the test data. The ratings are then linearly combined into final predictions based on the weights computed in the model blending part. Note that we do not retrain the weights of our blending model on the whole training set available to us but rather keep the ones found at the previous step.

L. Gridsearch summary

Here is a summary of the hyperparameters that are tuned for each algorithm, as well as the grid that is searched.

Model	Grid
Global Mean	N/A
User Mean	N/A
Movie Mean	N/A
Baseline	$\lambda \in \left\{10^i \, \forall i \in \{-8, ..., 3\}\right\}$
K-NN User	$k \in \{50, 100, 150, 200, 250\}$
K-NN Movie	$k \in \{50, 100, 150, 200, 250\}$
Slope One	N/A
SVD	$\lambda \in \left\{10^i \, \forall i \in \{-8, ..., 3\}\right\}$ $k \in \{1, 5, 10, 100\}$
SVD Baseline	$\lambda \in \left\{10^i \, \forall i \in \{-8, ..., 3\}\right\}$ $k \in \{20, 50, 100, 200, 300, 400\}$
SVD++	N/A
Blending	$\lambda \in \left\{10^i \, \forall i \in \{-8, ..., 3\}\right\}$

Table V: Summary of the grids that we searched in order to tune the selected hyperparameters

IV. Results

Our results are summarized in Table VI. Note that the RMSE for the blending algorithm was obtained on a subset of the validation set used for individual models. The ensemble method yielded the lowest RMSE on the crowdAI test set with a score of 1.018.

V. Discussion

We observe that every model outperforms the basic global mean solution. This is not surprising since these algorithms were heavily used during the Netflix prize [4]. Ultimately ensemble learning gives us the best results locally and in the crowdAI challenge. We would like to point out that better results could be achieved if we had access to additional metadata. Indeed SVD++ can be extended based on temporal information (timestamps) and additional implicit feedback (clicks, browsing history, etc). Moreover our grid search procedure could be more exhaustive with additional computational power at our disposal. This is especially true for time complex algorithms such as K-NN user based or SVD++ where hyperparameters tuning was constrained to small grids or not done at all. Besides neural networks models may improve the accuracy of our ensemble learning method as suggested in [5].

Model	RMSE	Optimal Hyperparameters	Weight
Global Mean	1.120	N/A	0.127
User Mean	1.096	N/A	-0.243
Movie Mean	1.030	N/A	-0.117
Baseline	1.003	$\lambda = 10^{-8}$	-0.023
K-NN User	0.992	$k = 250$	0.169
K-NN Movie	0.990	$k = 250$	0.154
Slope One	0.999	N/A	0.162
SVD	1.004	$\lambda = 0.01, k = 1$	-0.390
SVD Baseline	0.985	$\lambda = 0.1, k = 400$	1.033
SVD++	1.017	N/A	0.127
Blending	0.977	$\lambda = 10$	N/A

Table VI: RMSE score on the local validation set, optimal hyperparameters found through gridsearch as well as respective weights in the linear combination of the ensemble model

VI. Conclusion

In this paper we discussed and evaluated many models based on Collaborative Filtering for movie ratings prediction. We demonstrated the advantage of ensemble learning using a blend of 10 algorithms inspired by Matrix Factorization and K-NN techniques. The best individual model gave a RMSE of 0.985 on a validation set while the model blending algorithm gave a RMSE of 0.977. Ultimately we achieved a RMSE of 1.018 on crowdAI's EPFL ML Recommender System challenge.

REFERENCES

[1] crowdAI, "crowdai, epfl ml recommender system challenge," https://www.crowdai.org/challenges/epfl-ml-recommender-system/dataset_files, accessed: 10.12.2018.

[2] N. Hug, "Surprise, a Python library for recommender systems," http://surpriselib.com, 2017.

[3] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 471–475.

[4] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[5] M. Jahrer, A. Töscher, and R. Legenstein, "Combining predictions for accurate recommender systems," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 693–702.