Machine Learning Algorithms for Recommender Systems

Efe Acer, Daniil Dmitriev, Murat Topak
CS433 Machine Learning, EPFL, Project 2

***Abstract*—This paper aims to explore, compare and combine different algorithms that predict the ratings which will be given to movies by potential users, based on the patterns emerged in a user-movie rating dataset. To that end, we consider simple Baseline Models, user and item based Collaborative Filtering Algorithms and Matrix Factorization techniques and evaluate the performance of each in terms of Root Mean Squared Error. In brief, we find out that different algorithms capture different patterns in the dataset, hence we conclude that a combined model performs the best.**

I. Introduction

The ultimate goal of Recommender Systems is to assist users to access large digital collections in a personalized manner. They usually do this by predicting the order of preference, the rating, that a user would give to an item. The need for Recommender Systems increased greatly as a result of digital content providers, such as YouTube, Netflix and Amazon, becoming progressively widespread. Such digital content providers consistently seek for better Recommender Systems in order to provide better user experience.

The most naive way for a Recommender System to predict ratings is to make use of the core statistical properties of the dataset, such as the global mean of the ratings, mean of the ratings each user gave or mean of the ratings each item received. We categorize such simple models as **Baselines**.

Another set of models that Recommender Systems rely on is based on a technique called **Collaborative Filtering**. The underlying assumption of this technique is that if two users, say A and B, have given similar ratings to the movies they have rated in common; then it is more likely for A to have a similar opinion as B about any other movie than a randomly chosen user. The same logic applies for movies. This technique leads us to **Neighborhood Based Models**, where we need to define a similarity measure for users and items.

A newer and more popular set of models used in Recommender Systems stands on **Matrix Factorization**, which is a common technique to factorize a large matrix using two lower rank matrices. In our case, the goal is to represent users and movies in a lower dimensional latent space by learning two lower rank matrices that contain latent feature vectors for users and movies. This is done in a way that the dot product of a user feature vector and a movie feature vector recovers the rating that represents the user-item interaction.

The rest of this paper offers a methodology to combine these different techniques to capture all patterns in the dataset that are recognized by the individual models.

II. Data Exploration

A. Description of the Dataset

The dataset that we worked on consists of the ratings given to 1000 different movies by 10000 different users. The ratings are given in an integer scale from 1 to 5, 1 being the lowest order of preference and 5 being the highest. Since it is not practical for a user to rate each one of the movies, our training set only contains 1176952 ratings. This corresponds to a sparsity around 99.9% when we represent the training set as a matrix. Thus, our challenge is to learn 99.9% of the training matrix with the lowest possible Root Mean Squared Error (RMSE).

B. Analysis of the Dataset

Before implementing our machine learning procedure, we analyzed our dataset in order to make sure that the ratings were obtained from legitimate users. In other words, we made sure that there are no bots, spammers, etc. that generated uniform ratings. As it can be seen from Figure 1 below, the variance of user ratings form a Gaussian distribution, which ensures that there are no significant bots or spammers that should be taken care of.

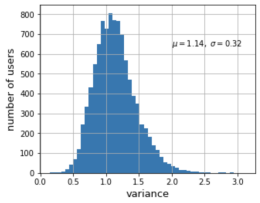

Fig. 1: Distribution of rating variances per user

Another important thing that we have checked is the deviation of ratings for users and items. We found out that some users tend to rate higher or lower, where similarly some movies tend to be rated higher or lower. This is a useful aspect of the dataset that led us to the BSGD model, which is explained in detail in the next section. Figure 2 shows the deviations:

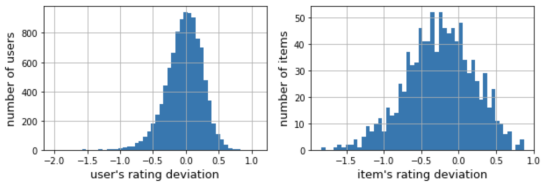

Fig. 2: Difference between user/item rating mean and the overall mean

We did no preprocessing seeing that it is not required.

III. Models

A. Baselines

1) Global Mean: The very first model uses all the ratings to compute the global mean and simply returns this value as the prediction. The global mean value also establishes a baseline score for the other models.

$$\hat{x} := \frac{1}{|\Omega|} \sum_{(n,d)\in\Omega} x_{nd} \tag{1}$$

where Ω is the set of observed entries in the training matrix.

2) User Mean: A slightly better model is to compute the mean for each user, then predict the corresponding mean for a specified user as the rating. The mean value of the ratings given by user n is formulated as follows:

$$\hat{x}_n := \frac{1}{|\Omega_{n:}|} \sum_{d\in\Omega_{n:}} x_{nd} \tag{2}$$

where $\Omega_{n:}$ is the set of movies rated by user n.

3) Item Mean: Another simple model is to compute the mean for each movie rather than each user. The mean value of the ratings given to movie d is given as follows:

$$\hat{x}_d := \frac{1}{|\Omega_{:d}|} \sum_{n\in\Omega_{:d}} x_{nd} \tag{3}$$

where $\Omega_{:d}$ is the set of users who rated the movie d.

B. Neighborhood Based Models

To have a neighborhood relation inside the dataset, we need to define a similarity metric. We chose to set the commonly used pearson correlation as our similarity metric, for two different users a and b the pearson correlation is given by:

$$p(a,b) = \frac{\sum_{i=1}^{N}(r_a(i)-\overline{r_a})(r_b(i)-\overline{r_b})}{\sqrt{\sum_{i=1}^{N}(r_a(i)-\overline{r_a})^2}\sqrt{\sum_{i=1}^{N}(r_b(i)-\overline{r_b})^2}} \tag{4}$$

where i gathers the movies rated by both a and b, $r_a(i)$ is the rating given by user a to movie i, and $\overline{r_a}$ is the average rating given by a (Same definitions apply for $r_b(i)$ and $\overline{r_b}$. The formulation is the same when a and b are movies and i is the set of common users.).

1) User kNN: With user-based kNN, given a rating x_{ad} to predict, we compute the k-nearest neighbor users of a, who have already rated d. Then we combine their rating deviations using the similarities as weights so that the most similar neighbors have more effect on the prediction. After that, we add the average rating given by a and output the result as the prediction. This procedure is formulated as follows:

$$\hat{r_a}(d) = \hat{r_a} + \frac{\sum_{b\in N(a)} \mathrm{sim}(a,b)(r_b(d)-\overline{r_b})}{\sum_{b\in N(a)} \mathrm{sim}(a,b)} \tag{5}$$

where $\mathrm{sim}(a,b)$ is the similarity of users a and b, defined by pearson correlation, $N(a)$ is the k-neighborhood set of a and d is a movie that is not rated by a. [1]

2) Item kNN: This model is analogous to the previous one. The only difference is that it runs over the similar *movies* instead of users. Since the number of movies is far fewer than the number of users, this model is computationally cheaper than the previous, hence it would be preferable in practice.

C. Matrix Factorization

Given N users, D movies, and the sparse matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$ representing the ratings, our goal is to construct a low rank approximation of $\mathbf{X}$ as the product of two matrices $\mathbf{X} \approx \mathbf{Z}\mathbf{W}^T$, where $\mathbf{Z} \in \mathbb{R}^{N\times K}$ and $\mathbf{W} \in \mathbb{R}^{D\times K}$ for some (small) number of latent variables K.

In this formulation, the matrix $\mathbf{Z}$ contains the latent feature vectors of the users as its row vectors and $\mathbf{W}$ does the same for the movies. To gain a better understanding of this formulation consider an element of a movie vector as the scariness of a movie. Similarly, consider an element of a user vector as the measure of user preference for scary movies. In this case, the natural estimation is the inner product of these vectors, which will produce high values if the particular interest of a user aligns with the corresponding features of a movie.

Finding the low rank approximation can be expressed as the following optimization problem:

$$\min_{\mathbf{Z},\mathbf{W}} \sum_{(n,d)\in\Omega} [x_{nd} - r_{nd}]^2 + \lambda_z \|\mathbf{Z}\|^2 + \lambda_w \|\mathbf{W}\|^2 \tag{6}$$

where r_{nd} is the predicted rating for user n and the movie d and defined as $r_{nd} := (\mathbf{Z}\mathbf{W}^T)_{nd}$. λ_z, λ_w are added to the formulation as regularization coefficients that avoid overfitting. Ω is again the set of observed entries.

1) Stochastic Gradient Descent (SGD): One way to find a solution to the optimization problem given in (6) is to use Stochastic Gradient Descent. This algorithm iteratively constructs a better low rank approximation until a predefined cutoff or a convergence threshold. The set of observed ratings, Ω is shuffled before each iteration to preserve the Stochastic nature of the algorithm and avoid converging to a local minimum. In each iteration, for each n and d in Ω, the algorithm computes the gradient of the loss function (6) with respect to $\mathbf{z_n}$ (nth row vector) and $\mathbf{w_d}$ (dth row vector), then updates them with a certain learning rate using the gradients. The gradient updates are as follows:

$$\begin{aligned} \mathbf{z}_n &\leftarrow \mathbf{z}_n + \gamma(e_{nd}\mathbf{w}_d - \lambda_z \mathbf{z}_n) \\ \mathbf{w}_d &\leftarrow \mathbf{w}_d + \gamma(e_{nd}\mathbf{z}_n - \lambda_w \mathbf{w}_d) \end{aligned} \tag{7}$$

where γ is the learning rate and the error e_{nd} is defined as $e_{nd} := x_{nd} - r_{nd}$.

2) Biased SGD (BSGD): This model extends the SGD algorithm, described above, by the additional use of biases b_{nd}. The bias value b_{nd} is a combination of three types of biases:

$$b_{nd} := \hat{x} + b_n + b_d \tag{8}$$

Here $\hat{x}$ is the global mean of the given ratings, which is the emprical expected value. b_n is the user baseline value that is the expected rating of user n minus the global mean $\hat{x}$. Analogously, b_d is the item baseline value giving us the expected rating of movie d minus the global mean. Hence, the summation b_{nd} gives us a baseline estimate for a user-movie rating.

Using b_{nd}, we modify the rating prediction r_{nd} as:

$$r_{nd} := b_{nd} + (\mathbf{Z}\mathbf{W}^T)_{nd} \tag{9}$$

Our optimization problem now becomes:

$$\min_{\mathbf{Z},\mathbf{W},\mathbf{b}_n,\mathbf{b}_d} \sum_{(n,d)\in\Omega} [x_{nd} - r_{nd}]^2 + \lambda_z\|\mathbf{Z}\|^2 + \lambda_w\|\mathbf{W}\|^2 + \lambda_n\|\mathbf{b}_n\|^2 + \lambda_d\|\mathbf{b}_d\|^2 \tag{10}$$

where $\mathbf{b}_n$ and $\mathbf{b}_d$ are the vectors containing user and movie biases.

We add two new update rules to (7) in order to learn the biases:

$$\begin{aligned} b_n &\leftarrow b_n + \gamma(e_{nd} - \lambda_n b_n) \\ b_d &\leftarrow b_d + \gamma(e_{nd} - \lambda_d b_d) \end{aligned} \tag{11}$$

The intuition behind BSGD approach is that some users tend to rate movies overall higher than the others, and similarly, some movies are generally getting better ratings. This introduce some bias in the prediction that is not specific to the user-movie pair and needs to be excluded from the factorization.

3) Alternating Least Squares (ALS): Our optimization problem in (6) is a non-convex minimization, meaning that we cannot obtain an explicit solution. However, using a coordinate descent approach we can transform this non-convex problem into two separate convex problems by fixing either $\mathbf{Z}$ or $\mathbf{W}$. ALS algorithm makes use of this and iteratively solves for an optimal $\mathbf{Z}^*$ fixing $\mathbf{W}$ and an optimal $\mathbf{W}^*$ fixing $\mathbf{Z}$ using the well-known least squares technique. This procedure converges after some number of iterations. The normal equations are given below:

$$\begin{aligned} \mathbf{Z}^{*T}_n &= (\mathbf{W}^T\mathbf{W} + \lambda_z\mathbf{I})^{-1}\mathbf{W}^T\mathbf{x}_n \\ \mathbf{W}^{*T}_d &= (\mathbf{Z}^T\mathbf{Z} + \lambda_w\mathbf{I})^{-1}\mathbf{Z}^T\mathbf{x}_d \end{aligned} \tag{12}$$

where $\mathbf{x}_n$ is the nth row vector and $\mathbf{x}_d$ is the dth column vector of the rating matrix.

D. Others

1) SlopeOne: This model calculates prediction r_{nd} as:

$$r_{nd} = \hat{x}_n + \frac{1}{|R_d(n)|} \sum_{m\in R_d(n)} \text{dev}(d,g) \tag{13}$$

where $R_d(n)$ is the set of relevant items, meaning the set of items g rated by n that also have at least one common user with d. dev(d,g) is defined as the average difference between the ratings of d and those of g:

$$\text{dev}(d,g) = \frac{1}{|U_{dg}|} \sum_{n\in U_{dg}} r_{nd} - r_{ng} \tag{14}$$

This technique is simple yet useful. [2]

2) CoClustering: In this model, users and items are assigned some clusters C_n, C_d, and some co-clusters C_{nd}.

The prediction r_{nd} is set as:

$$r_{nd} = \overline{C_{nd}} + (\hat{x}_n - \overline{C_n}) + (\hat{x}_d - \overline{C_d}) \tag{15}$$

where $\overline{C_{nd}}$ is the average rating of co-cluster C_{nd}, $\overline{C_n}$ is the average rating of n's cluster, and $\overline{C_d}$ is the average rating of d's cluster. If the user is unknown, the prediction is $r_{nd} = \hat{x}_d$. If the item is unknown, the prediction is $r_{nd} = \hat{x}_n$. If both the user and the item are unknown, the prediction is $r_{nd} = \hat{x}$.

Clusters are assigned using a straightforward clustering algorithm such as k-means. [2]

E. Implementation of the Models

We used our own implementations for the Baseline models and Matrix Factorization models [3]. We particularly preferred to implement Matrix Factorization models ourselves, since their implementation significantly change from library to library.

For the Neighborhood Based Models and models explained in the *Others* section, we used the Surprise Library, written by Nicolas Hug. Since these models are computationally expensive, the cython (optimizing C based static compiler for Python) powered implementations of the Surprise Libray helped us a lot.

We used 5-fold cross validation to tune our hyperparameters, however because of the long runtimes of the models we could not spend much time cross validating.

IV. Blending

Ensemble learning methods use multiple models to obtain better predictive performance than the performance obtained from any of the constituent model alone [4]. Optimizing the weighted average of the models is one of the simplest ensemble methods that results in a blended (combined) model with better performance. Thus, we established our final model using a weighted average of the models explained in the previous section.

We used the Sequential Least Squares Programming (SLSQP) Method to find the optimal weights for our blended model, since it is a relatively fast method to minimize complicated functions. We defined the objective function as the test RMSE of our weighted model. For the implementation, we preferred the SLSQP method that is available as an option in `scipy.optimize` module's `minimize` function.

The individual test RMSEs, hyperparameters and blending weights of each constituent model together with the test RMSE of the blended model are given in table 1:

Model	test RMSE	blending weight	hyperparameters
Global Mean	1.12175	0.40429	
User Mean	1.09639	-0.42286	-
Movie Mean	1.03088	-0.31372	-
MF SGD	1.00171	-0.17466	# features $= 20$ # epochs $= 20$ $\lambda_z = \lambda_w = 0.07$ $\gamma = 0.02$
MF BSGD	0.99392	0.34547	# features $= 20$ # epochs $= 20$ $\lambda_z = \lambda_w = 0.07$ $\lambda_n = \lambda_d = 0.001$ $\gamma = 0.02$
MF ALS	0.98278	0.73774	# features $= 8$ # epochs $= 25$ $\lambda_z = \lambda_w = 0.081$
User kNN	0.99760	0.25695	$k = 100$
Item kNN	0.98662	0.34782	$k = 300$
SlopeOne	0.99919	-0.17666	-
CoClustering	1.01742	0.00329	# epochs $= 20$ # user clusters $= 10$ # item clusters $= 10$
Blending	0.97618	-	-

Table I: test RMSEs, blending weights and hyperparameters of each model (test set is a local validation set that is randomly splitted from the dataset with a ratio of 0.1)

V. Discussion

We have also implemented one more powerful model, namely *SVD++*. The work of Yehuda Koren shows that this model can obtain very good results [5] but due to insufficient computational power and time constraints, we did not succeed in adding it to our combined model. For the sake of completeness, we provide a small explanation of this model:

SVD++: In real platforms the users leave some implicit feedback information for the Recommender Systems, such as browsing data and history data. Our dataset only contains explicit information, however it can be argued that the fact that a user rated a movie provides an implicit feedback since it means the user has indeed watched the movie. Such an information might be useful to prevent a Recommender System from recommending the second film in a Saga to a user who did not watched the first film yet. Therefore, the SVD++ model introduces a factor vector ($\mathbf{y_g} \in \mathbb{R}^K$) for each item, and uses it to describe the implicit characteristics of the item. The predictive rating of the SVD++ model is

$$r_{nd} = b_{nd} + \mathbf{w}_d^T \left(\mathbf{z}_n + |R(n)|^{-1/2} \sum_{g \in R(n)} \mathbf{y_g} \right) \qquad (16)$$

where $R(n)$ is the set of movies rated by n and $|R(n)|^{-1/2}$ is just a normalizer.

We considered Koren's *integrated model* as well. This model is highly sophisticated, it introduces neighborhood information and redefines the predictive rating of SVD++ [6]. Again, we did not succeed adding it to our blend because of practical reasons.

There were also models that we implemented but passed over because of the fact that they did not improve the predictive performance of the blended model. Some of these models are global median, user median, movie median and Surprise Library's BaselineOnly model.

VI. Results and Conclusion

The results we have obtained throughout our work were in accordance with our theoretical expectations. Our combined model scores a RMSE of 1.017 on the crowdAI platform, which is the second best predictive performance. Given additional computational power, we believe that a new blend including the models mentioned in the previous section would achieve a much better performance.

Our work showed us how effective ensemble learning is. Each individual model we tried captures a different aspect of the dataset, i.e. Neighborhood models capture similarity relations, Matrix Factorization based models discover the underlying latent features and Baseline models capture the core statistical properties. Thus, a combined model allows for a much more flexible model that can make use of various aspects of the dataset.

References

[1] A. Kose, C. Kanbak, and N. Evirgen, "Performance comparison of algorithms for movie rating estimation," *CoRR*, vol. abs/1711.01647, 2017. [Online]. Available: http://arxiv.org/abs/1711.01647

[2] N. Hug, "Surprise, a Simple Recommender System Library for Python," 2016, [Online; accessed 18-December-2018]. [Online]. Available: http://surpriselib.com/

[3] GitHub, "EPFL_ML_Project2 Repository," 2018. [Online]. Available: https://github.com/efeacer/EPFL_ML_Project2

[4] Wikipedia, "Ensemble Learning — Wikipedia, The Free Encyclopedia," 2018, [Online; accessed 18-December-2018]. [Online]. Available: https://en.wikipedia.org/wiki/Ensemble_learning

[5] Y. Koren, "Collaborative filtering with temporal dynamics," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 447–456. [Online]. Available: http://doi.acm.org/10.1145/1557019.1557072

[6] ——, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08. New York, NY, USA: ACM, 2008, pp. 426–434. [Online]. Available: http://doi.acm.org/10.1145/1401890.1401944