An Implementation for Recommender System with Collaborative Filtering

Futong Liu, Neeraj Yadav, Zhechen Su

***Abstract*—Precise predictions of user preferences are of great significance of recommender systems performance. Provided with a data set of sparse ratings on items from different users, this paper discusses and compares possible solutions for such a prediction problem, and then proposes a methodology with a RMSE of 1.02. This methodology can be further refined by tuning the model more precisely or importing more models.**

I. Introduction

Recommender systems have matured prevailing and ubiquitous in various fields such as online retailers, streaming content providers and online dating. Recommender systems predict the ”rating” or ”preference” a user would give to an item based on the assumption that there are clusters of users who behave in a similar way and have comparable needs and preferences.

Generally, there are two distinct approaches of recommender systems: content-based filtering and collaborative filtering. In contrast to content-based filtering, which uses a series of discreet indexes to characterise an item, collaborative filtering requires a considerable amount of data to predict users' preference [1].

Collaborative filtering can then be formulated as memory-based or model-based. The memory-based method computes the similarities among neighbours in memory by using the known ratings of users. It is so called memory-based, as the database of entries is loaded into memory and used directly to generate a recommendation. On the other hand, the model-based approach derives a model utilizing the available data offline, where predictions are made based on this model, and the recommendation process takes place online instantaneously [2].

The objective of the project is to develop and refine an implementation for a recommender system with decent prediction accuracy measured by RMSE using collaborative filtering technique. Accordingly, 10 models are implemented ultimately, eventually achieving a RMSE of 1.02 on CrowAI, the Recommender System Challenge of EPFL.

This paper first analyses the data set, and then introduces the model to be implemented. Finally, the results of these models are summarised, compared and concluded.

II. Exploratory Data Analysis

A. Data Set

For the project, a data set is given that consists of ratings of 10000 users for 1000 different items. All ratings are integer values between 1 and 5 stars and no additional information is available on the movies or users. This is a relatively sparse data set, as there are only $1,176,952$ non-zero entries, accounting to nearly 12% of the whole.

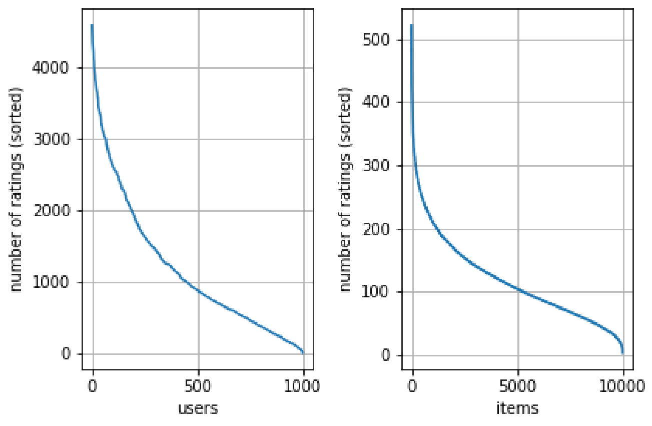

Fig. 1. The number of ratings per user and per item

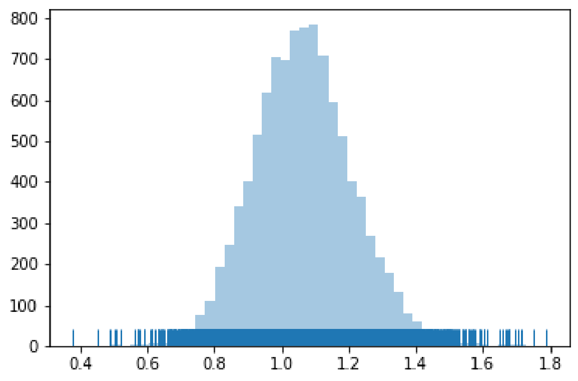

Fig. 2. Variance distribution of users

B. User Participation

Fig. 1 shows the number of ratings per user and per item respectively. It can be concluded that this data set in in good quality with respect to user participation: most users give a good amount of ratings to items and most items are rated by users with a median of about 100.

Additionally, the most inactive user gives ratings to 8 items in total, while the most unpopular item only has 3 ratings available. These inactive users and unpopular items, of which the sample space is too small to generate a suggestive model for prediction, therefore have to be filtered out from the training set with a threshold of 10.

C. User Variance

As in Fig. 2, the variance distribution of users' ratings is depicted. It is observed that it approximately follows a Gaussian distribution and hence the data set is reliable to use. All users are consistent with their ratings, since no user gives a uniform rating to all its rated items and no biased user gives either 1 or 5.

III. Models and Methods

A. Baseline Models

There are several baselines implemented to be compared as the baselines.

The global mean:

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

where Ω comprises the non-zero rating indexes (d,n) in the training data set.

The user mean:

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

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

The item mean:

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

where $\Omega_{d:}$ is the set of users who have rated the d-th item.

B. Matrix Factorization with SGD

Given the data set $\mathbf{X} \in \mathbb{R}^{D\times N}$, matrix factorization aims to find two thin matrix, namely the item matrix $\mathbf{W} \in \mathbb{R}^{D\times K}$ and the user matrix $\mathbf{Z} \in \mathbb{R}^{N\times K}$, where $K \ll D, N$ is the number of latent variable dimensions (features). Qualitatively, $\mathbf{W}$ depicts the main characteristics of each item by K features, while $\mathbf{Z}$ similarly represents those of each user by K features. Therefore, those missing ratings can be predicted by the inner product:

$$\hat{X} \approx \mathbf{W}\mathbf{Z}^T \tag{4}$$

In order to minimize the RMSE while avoiding overfitting, a regularized loss function is used:

$$L(\mathbf{W},\mathbf{Z}) = \frac{1}{2} \sum_{(d,n)\in\Omega} [x_{dn} - (\mathbf{W}\mathbf{Z}^T)_{dn}]^2 + \frac{\lambda_{it}}{2}\|\mathbf{W}\|^2_{Frob} + \frac{\lambda_{us}}{2}\|\mathbf{Z}\|^2_{Frob} \tag{5}$$

where

$$f_{dn} = \frac{1}{2}[x_{dn} - (\mathbf{W}\mathbf{Z}^T)_{dn}]^2 \tag{6}$$

is the (d,n) term of the sum over all the valid ratings which the cost function consists of, and λ_{it} and λ_{us} are the penalization coefficients.

The penalization coefficients λ_{it} and λ_{us} are optimized by cross validation.

Stochastic Gradient Descent (SGD) computes the gradient at a randomly-chosen fixed element (d,n) with respect to $\mathbf{W}$ and $\mathbf{Z}$ respectively:

The update rule:

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \gamma\nabla_{\mathbf{W}} f_{dn} \tag{7}$$

$$\mathbf{Z}^{(t+1)} = \mathbf{Z}^{(t)} - \gamma\nabla_{\mathbf{Z}} f_{dn} \tag{8}$$

C. Matrix Factorization with ALS

With the same rationale of matrix factorization, an alternative approach other than SGD is to solve the normal equations to find the optimal $\mathbf{W}$ and $\mathbf{Z}$.

By setting to zero the gradient of RMSE with respect to Z with W fixed, it is derived that

$$\mathbf{Z}^{*T} = (\mathbf{W}^T\mathbf{W} + \lambda_{us}\mathbf{I}_K)^{-1}\mathbf{W}^T\mathbf{X} \tag{9}$$

Analogously, equating to zero the gradient of RMSE with to respect to W with Z fixed, it is derived that

$$\mathbf{W}^{*T} = (\mathbf{Z}^T\mathbf{Z} + \lambda_{it}\mathbf{I}_K)^{-1}\mathbf{Z}^T\mathbf{X}^T \tag{10}$$

where λ_{us} and λ_{it} are the regularisation parameters, which ensures the matrix to be inverted is not singular.

With the crucial equations derived, the procedure of ALS is briefed as below:

- Initialize $\mathbf{W}$ and $\mathbf{Z}$
- Iterate according to the stop criteria
  - Update $\mathbf{Z}$ according to equation (9)
  - Update $\mathbf{W}$ according to equation (10)
- Compute $\mathbf{X} = \mathbf{W}\mathbf{Z}^T$

D. Memory-Based Collaborative Filtering

Memory-based collaborative filtering algorithms, also referred to as neighborhood-based algorithms, were among the earliest algorithms developed for collaborative filtering [2]. These algorithms are based on the fact that similar users display similar patterns of rating behavior and similar items receive similar ratings. KNN is such a prediction algorithm that computes a prediction for the rating exploiting a weighted sum of the other users/items ratings with the help of a similarity metric, in our case Pearson Baseline. This algorithm implemented in the Python Surprise library [3].

E. Multi-layer Perceptron

A neural network is implemented with an embedding layer to learn the representation of movie and user and 4 fully connected layers. The architecture is ameliorated by varying the number of neurons in the fully connected layers and depth of the layers. The fully connected layers are followed by the activation function. Sigmoid activation function gets saturated while learning, and hence Relu activation function is used. Furthermore, as the architecture depth increases, the model overfits and under-performs for the validation set, therefore dropout layers are used between the consecutive fully connected layers to create independencies between the inputs that are fed to the fully connected layers. An optimal probability is set for dropping the inputs. Categorical Cross Entropy and Mean squared Error are implemented for computing the costs. For optimizing the loss function, Adam and SGD optimizer are implemented. Experiments are made based on varying the learning rate, the number of epochs and batch size to achieve optimal hyper parameters. The architecture is inspired by [4].

F. Singular Value Decomposition

Standard SVD is implemented to reduce the number of features of a data set. It decomposes the data set $\mathbf{X} \in \mathbb{R}^{D\times N}$ as below:

$$\hat{X} \approx \mathbf{WSZ}^T \tag{11}$$

where matrix $\mathbf{W} \in \mathbb{R}^{D\times K}$ is a left singular matrix, representing the relationship between users and latent factors. $\mathbf{S} \in \mathbb{R}^{K\times K}$ is a diagonal matrix describing the strength of each latent factor, while $\mathbf{Z} \in \mathbb{R}^{N\times K}$ is a right singular matrix, indicating the similarity between items and latent factors.

In vector form, each item is represented by a vector q_i and user by a vector p_u.The dot product of those 2 vectors is the expected rating

$$\hat{r}_{ui} = {q_i}^T p_u + b_u + b_i + \mu \tag{12}$$

where μ is the average ratings by users, b_i is the bias of item and b_u is the bias of user

In order to avoid over-fitting, regularized squared loss is used:

$$\sum_{(u,i)\in\Omega} [r_{ui} - \hat{r}_{ui}]^2 + \lambda({b_u}^2 + {b_i}^2 + \|q_i\|^2 + \|p_u\|^2) \tag{13}$$

The updating rule is as below:

$$b_u = b_u + \gamma(e_{ui} - \lambda b_u) \tag{14}$$

$$b_i = b_i + \gamma(e_{ui} - \lambda b_i) \tag{15}$$

$$p_u = p_u + \gamma(e_{ui}{\cdot}q_i - \lambda p_u) \tag{16}$$

$$q_i = q_i + \gamma(e_{ui}{\cdot}p_u - \lambda q_i) \tag{17}$$

where $e_{ui} = r_{ui} - \hat{r}_{ui}$

G. Ensemble

According to the paper [5] explained by team *Bellkor's Pragmatic Chaos*, the winner of 2009 Netflix Prize, its solution was optimized by blending their numerous different models. Though not having as many models as them, it is nevertheless inspiring and feasible to blend the models implemented above to generate a new set of predictions.

Hence, a simple linear model is trained to map the outputs of these implemented models to a single prediction. Mathematically, the final prediction is a linear combination of the results of each model. The goal is to find the least squares problem to minimize the RMSE of $\hat{\mathbf{Y}}$:

$$\begin{bmatrix} \hat{r}_{1,1} & \cdots & \hat{r}_{1,10} \\ \vdots & & \vdots \\ \hat{r}_{N,1} & \cdots & \hat{r}_{N,10} \end{bmatrix} \mathbf{w} = \hat{\mathbf{X}}\mathbf{w} = \hat{\mathbf{Y}}$$

where matrix $\hat{\mathbf{Y}}$ consists of 10 columns, where each item $\hat{r}_{i,j}$ is the estimation of the i-th rating by the j-th model. Vector $\mathbf{w}$ contains the weights for each model. The final prediction $\hat{\mathbf{Y}}$ is a weighted average of the predictions of each model. This is a regression problem that can be solved using penalization on $\mathbf{w}$ and then solving a ridge regression problem.

IV. Results and discussion

A. Summary of Test Results

Model	Parameters	RMSE
Global Mean	-	1.1183
User Mean	-	1.0289
Item Mean	-	1.0938
MF-SGD	$\lambda_{it} = 0.25$, $\lambda_{us} = 0.01$, $k = 20$	1.0001
MF-ALS	$\lambda_{it} = 0.575$, $\lambda_{us} = 0.014$, $k = 20$	0.9839
SVD	latent factor=105, epochs = 25	0.9958
SVD++	latent factor=22, epochs = 15	0.9960
KNN-item based	similarity='msd', $k = 20$, $min_k = 1$	1.0234
KNN-user based	similarity='msd', $k = 80$, $min_k = 1$	1.0196
Multi-layer Perceptron	lr = $10^{-4}, batchsize = 256, epochs = 25$	0.9980
Best Model	MF-ALS	0.9839

TABLE I
Benchmark of implemented models

The RMSE is tested on the test set of the data set, with a split ratio of 10%.

B. Baselines

Although the baseline algorithms are simple and easy, but their performance is surprisingly not bad, especially the user Mean. This is because most users are persistent with their ratings, as explained in section II-C, in a specific range of variance distribution. The run time of global mean is 25s, with user mean 167s and item mean 32s.

C. Matrix Factorization with SGD

In order to tune the hyper-parameters for matrix factorization with SGD, a k-fold cross validation is used, with k set to be 5. By using grid search, the item penalization coefficient $\lambda_{it} = 0.25$, the user penalization coefficient $\lambda_{us} = 0.01$, the latent variable $k = 20$. This process of SGD is iterated by 50 times, when the change between iterations are small enough to be ignored. The running time of this method is 2067s.

D. Matrix Factorization with ALS

With the same idea as SGD, matrix factorization with ALS is tuned by using grid search as well. After setting $k = 20$, which is found at the initial cursory grid search, it is observed that most results of ALS are better than SGD. Hence, its parameter optimization is investigated more precisely with a finer grid in grid search. Fig 3 below shows the grid search plot, where the brightest area indicates the most precise prediction. The best-tuned model found turns out to have the item penalization coefficient $\lambda_{it} = 0.575$, the user penalization coefficient $\lambda_{us} = 0.014$. The model is trained such that the change of improvement between each iteration is neglectable (10^{-6}). The running time of this method is 1847s.

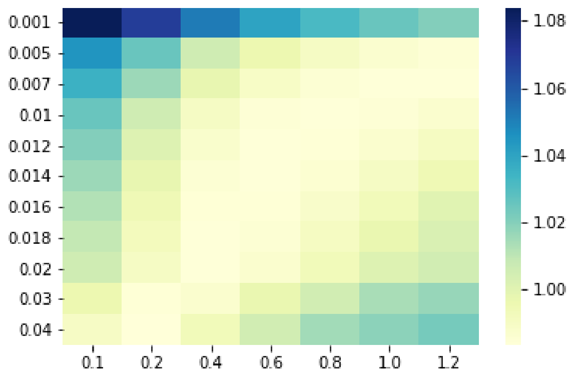

Fig. 3. Grid search plot for optimising MF-ALS with $k = 20$

E. Memory-Based Collaborative Filtering

$min_k(int)$ is the minimum number of neighbors to take into account for aggregation. If there are not enough neighbors, the prediction is set the the global mean of all ratings. As default, $min_k(int)$ is set to 1. Additionally, by experiments, for user-based KNN, k is set to 80, while for item-based KNN, k is set to 20.

F. SVD and SVD++

The hyper-parameters of SVD were tuned using grid search and the model gave optimum rmse when ran for 25 epochs with 105 latent factors and regularization of 0.05. The SVD++ gave best result with 22 latent factors, 15 epochs and regularization of 0.04.

G. Multi-layer Perceptron

The Neural network is optimized with Adam optimizer in 26 epochs with a learning rate of 0.0001 and batch size of 256. It was observed that loss function for validation data fluctuates very rapidly for higher learning rates and smaller batch size, as shown in Fig. 4. The model overfits for epochs greater than 26. A further increase in the number of neurons in the layers increases the overhead of parameters to be learned, negatively influencing the capability of model. Dropout probabilities between the layers is set between 0.1 to 0.25. If dropout probabilities are further increased then model doesn't learn efficiently and training loss decreases significantly.

H. Ensemble

In the definitive comparison at section IV, matrix factorization with ALS proves to be the best with the smallest RMSE.

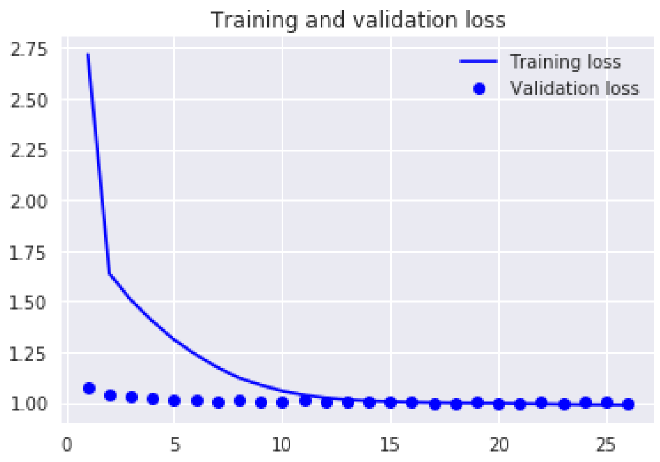

Fig. 4. Training Curve for Multi-layer Perceptron

By blending the models together, the best results are computed. The weights are selected and optimised by using ridge regression. However, the best result turns out to be with a single scaling number of 1.02774082 on the model MF-ALS while other weights turn out to be 0, meaning that this is simply a weighted version of the results of ALS itself, which contradicts the initial guess inspired by the literature [5]. This strange result is probably because one or more models are not tuned accurately, which sways the final weighted model and violates the ensemble approach. The final model used for submission is the pure MF-ALS with all its elements multiplied by 1.02774082.

V. Conclusion and Future Work

Under the umbrella of collaborative filtering, totally 11 methods are implemented and compared, among which a weighted version of matrix factorization with ALS prevails over others. It is concluded that ALS and SGD performs differently even if they intend to minimize exactly the same cost function. Additionally, the selection of parameters for the very same model holds sway over its performance.

However, there are also drawbacks for the implementation of this paper, based on which further work and improvement is possible. This include adding more methods to the ensemble, or even the same model but with different parameters. Some models can be optimized further with stronger computation power. It is also probable to implement a shallow and deep neural net for different embedding layers separately and then taking their dot product.

References

[1] Claudio Adrian Levinas. An analysis of memory based collaborative filtering recommender systems with improvement proposals. Master's thesis, Universitat Politècnica de Catalunya, 2014.
[2] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer, Cham, 2016.
[3] Surprise, a simple recommender system library for python.
[4] N. Wang H. Wang and D.-Y. Yeung. Collaborative deep learning for recommender systems. 09 2015.
[5] Yehuda Koren. The bellkor solution to the netflix grand prize. 09 2009.