Machine Learning Project 2 - Recommender System

Yawen Hou, Peilin Kang, Yueran Liang
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—This project aims to implement a recommender system using different models including baselines, KNN, SVD, and ALS, some of which came from some machine learning packages and some of others were written by us. The method we use to evaluate the models is RMSE. At last, the results all the models are stacked together with optimized weights to reach the lowest RMSE.**

I. Introduction

Collaborative filtering is a technique used by recommender systems to predict the preferences of users in order to make them good recommendation. Multiple algorithms can achieve this; in this project, we used different models from different packages. These algorithms include: baselines predictors wrote by ourselves; from Surprise [1]: singular value decomposition (SVD), SVD++, an SVD that takes into account implicit ratings (an implicit rating solely assumes the fact that a user rated an item, regardless of the rating value) and K-Nearest neighbour(KNN); from Spotlight [2]: an implementation of matrix factorization called Explicit Factorization; from PyFM: [3]: a factorization machine implemented on pyTorch called pyFM; from sklearn: an implementation of matrix factorization using alternating least square(ALS) and ridge regression(MF RR). After tuning hyper-parameters for each model using 5-fold cross-validation, we decide to imitate the Netflix winner and mix the results of our models using stacking to output our lowest prediction.

II. Data Exploration

The dataset contains two columns, showing in order the ID (user/movie ids) and the rating of the specific user for the specific movie. The ratings are scaled on an integer scale from 1 to 5. If we put that data into a matrix, then the ID also represent the row id (user) and the column id (column) of the corresponding rating. There are 10,000 users and 1,000 movies in total. The number of ratings is $1,176,952$, which means that non-zero entries only occupy about 12% of sparse matrix with dimensions $10,000 \times 1,000$.

To illustrate our sayings, we also plot the distribution of number of ratings per movie and ratings per user to see if there exist a great amount of inactive users or invalid movies. An inactive user designate a user that is registered but never vote any movies; it would therefore be impossible to predict the preferences of this kind of users since we don't have any data for them. Similarly, if we don't have any rating for a particular movie, it would be difficult to evaluate the rating of any user for this movie. From the figure 1 and figure 2, we can see that the number of inactive users and invalid movies is very low - this will not impact the precision of algorithms. Therefore, we decided to not filter out any users/movies and proceed with the whole data.

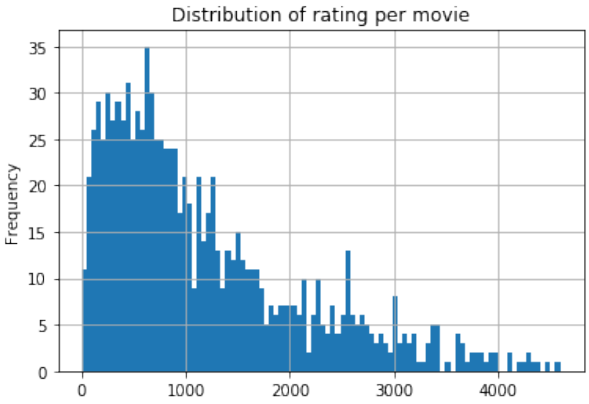

Fig. 1: Distribution of number of ratings per movie

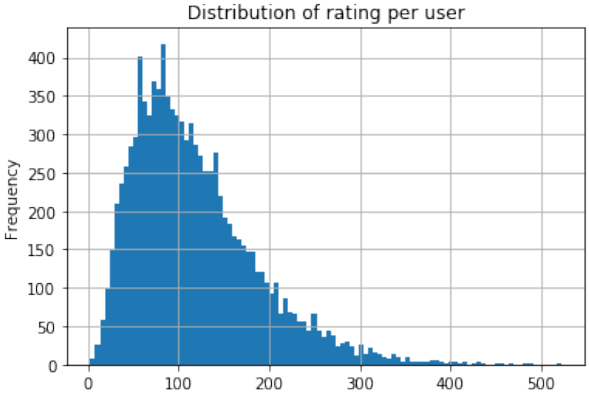

Fig. 2: Distribution of number of ratings per user

III. Models Description

A. Baseline Predictors

First, we implemented the baseline predictors using the average/ median of global rating and the average/median rating of each user or movie. In short, we calculate the mean/median of all the users and movies; of the users only; and of the movies only and set those means(s)/median(s) as the predicted rating. It can help us find the user bias and movie bias and discover the underlying tendency. More importantly, it provides basic figures for us to evaluate our advanced models afterwards.

From the previous step, we found that large users' biases and movies' biases exist. The user's bias is also called user's

habit (or user's mood) [4], which indicate the rating habits of each user. For example, if Joe is a critical user, he tend to give lower rating than others on each movies. On the other hand, the movie's bias shows the relatively quality. For instance, Titanic is a good movie so its rating tend to be better than others. These two factors are taken this factor into account to build a general baseline predictor:

$$\begin{aligned}\hat{p}_{u,m} &= \mu + b_u + b_m \\ &= \mu + \mu_u - \mu + \mu_m - \mu \\ &= \mu_m + \mu_u - \mu\end{aligned}$$

where μ is the global mean, μ_u is each user's mean and μ_m is each movie's mean.

In addition, we do not want the prediction is much influenced by the difference of users' habits, so we implemented the functions that standardize user preferences and movie biases (see baseline.py) to get rid of the influences of the biases when we analyzes the labelled data (data-train.csv), and then recover from the standardization after outputing the predictions. This showed to have lowered by a little our errors, but not much.

B. PyFM

PyFM is a Factorization Machine algorithm implemented with Python using stochastic gradient descent (SGD) with adaptive regularization as a learning method, i.e. in short, it is an implementation of matrix factorization using SGD. We decided to use the method in the hope to capture different features of the users/movies than the other models. The underlying theory is similar to the matrix factorization using ALS in section G.

C. Singular Value Decomposition

The Singular Value Decomposition uses the same equation as explained in class, implemented by the Surprise library:

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

where rui is the estimation for user u and item i, bu is the user bias, bi is the item (movie) bias, qi is the the vector representing the features of the item i, and pu is the vector representing the features of user u (which norms we need to minimize). The minimization process goes as following:

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

The particularity of this algorithm is that it let us tune the learning rate for the user parameters (bu, pu) and for the movie parameters (bi, qi) separately. Therefore, we have more control and the results should be more precise.

D. SVD++

This is an extension of the SVD algorithm taking into account implicit ratings. An implicit rating takes into other hidden parameters than just the score number; for instance, the number of times the user watched a movie could be taking into account as a hidden variable hinting the latter's preference. Although in our case, we did not have any hidden parameters, we decided to try this model in order to see if we could capture some other things than just using svd. The equation of the model is the following:

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

where the yj terms are the factors used to capture implicit ratings. The rest of the parameters are the same as what has been explained above in the SVD section.

E. K-Nearest Neighbour

KNN is an algorithm that predicts the rating for a user-movie pair by consulting k number of similar pairs around them. The model provided by Surprise performs a weighted sum of the other users/movies ratings where the weights are determined proportionally to a similarity metric. Here, we could choose the metric and the number of nearest neighbour, and after tuning it turns out that Pearson Baseline and k of 200 give the best results.

F. Spotlight Explicit Factorization

Spotlight provide implementations of recommender models using PyTorch. The explicit factorization model is a classic matrix factorization, with latent vectors used to represent both users and items. Their dot product gives the predicted score for a user-item pair. The reason we chose this library is the various hyperparameters different from other models that we could choose to tune, for instance the loss parameter - we could choose one of regression, poisson, logistic, to calculate our loss, the L2 loss penalty (float), and the batch_size (for internal cross-validation). In fact, this is one of the few models that gave us a RMSE of higher than 1.025 when used alone.

G. Matrix Factorization using Alternating Least Square

The Matrix factorization using alternating least square is an adapted implementation of the equations given in the lecture note. We factorize the original X matrix into two feature matrices with K latent factors, one for users $W \in R^{K \times D}$ and the other for movies $Z \in R^{K \times N}$. In addition, two regularizers are also added for each feature matrix with λ_w for user and λ_z for movie to avoid over-fitting. And because these two matrices are sparse, it is better to multiply each regularizer with the number of non-zero entry w_d for each user and z_n for each movie and before summing them up. The final cost function is represented as the following:

$$\begin{aligned}L =& \frac{1}{2} \sum_{(d,n) \in \Omega} [x_{dn} - (WZ^T)_{dn}]^2 \\ &+ \sum_{d \in D} w_d \times \frac{\lambda_w}{2} \|W_{:d}\|^2 + \sum_{n \in N} z_n \times \frac{\lambda_z}{2} \|Z_{:n}\|^2\end{aligned}$$

where $\Omega = [D] \times [N]$.

Coordinate decent is used to minimize the cost function. We first minimize W for fixed Z and then minimize Z given W. Again, due to the sparsity of the original matrix, we cannot update the whole matrix at once (referring to the equation in the ALS section of lecture notes, which assume that we have

a dense matrix). Therefore, the user feature matrix is updated one user at a time, and so is the movie feature matrix. In short we update each user/movie feature as the following:

For each user,

$$W_d = (ZZ^T + w_d \times \lambda_w I)^{-1} Z X_{d:}$$

For each movie,

$$Z_n = (WW^T + z_n \times \lambda_z I)^{-1} W X_{:n}$$

H. Matrix Factorization using Ridge Regression

The principle of this model is similar to the ALS. However, instead of computed the W and Z using scipy sparse matrix manually, these two matrices is updated by the ridge regression model provided in `sklearn.linear_model` python package.

IV. Models' Parameters Tuning

A 5-fold cross-validation using grid search is implemented to tune the hyper-parameters for most of the models. Exceptionally, for ALS, we chose to use random search [5] to make the tuning process more efficiently as it has too many parameters to tune. In a limited time, here are the best parameters for each model we got: Table I.

Model	Parameters	#Factors
SVD	lr_bu=1e-9 , lr_qi=1e-5	n_factors =50
	reg_all=0.01	
	n_epochs=200	
KNN	k=100,	
	sim_options= {'name': 'pearson_baseline', 'user_based': False}	
SVD++	lr_bu=1e-9 , lr_qi=1e-5	n_factors =50
	reg_all=0.01	
	n_epochs=200	
Spotlight	loss='regression'	embedding_dim =150
	batch_size=256	
	l2=1e-5	
	learning_rate=0.0001	
	n_iter=50	
ALS	weight = 2.18644068	num_features = 20
	lambda_movie = 0.02	
	lambda_user = 0.47	
	iterations=50	
PyFM	verbose=True	num_factors =20
	task="regression"	
	initial_learning_rate=0.001	
	learning_rate_schedule= "optimal"	
	num_iter=200	
MFRR	alpha=19	num_features =20
	iterations=50	

TABLE I: Parameters for each model

V. Model Analysis and Blending

We optimized our prediction with combination of all the models and gave each a weight to create the final solution. Models are blended using linear regression to fit and get weights from the regression result. The original RMSE of each model and their wights after blending is shown in Table II where it is obviously seen that the RMSE dropped significantly after combination.

Model	RMSE	Weight
global mean	1.127	0.043
global median	1.127	0.043
movies' mean	1.07	0.009
movies' median	1.098	0.050
users' mean	1.117	-0.052
users' median	1.147	-0.005
movies' mean_user_std	1.046	0.141
movies' median_user_std	1.067	0.150
movies' median_user_habit	1.088	-0.045
movies' median_user_habit_std	1.064	-0.118
general baseline	1.046	0.086
general baseline_user_std	1.046	-0.118
SVD	1.025	0.011
KNN	1.025	0.030
SVD++	1.025	0.009
Spotlight	1.022	0.289
PyFM	1.025	0.282
ALS	1.031	0.215
MF RR	1.04	0.167
Blending	1.017	

TABLE II: Models' RMSE and weights

VI. Result and Conclusion

By blending the model and using the best parameters we tune, the best RMSE we got in CrowAI is 1.017. Actually for each model, RMSE are lower than 1.0 when we tested them in our local machine most of the time. It may be because our model is a little over-fitting or the valid type of rating in CrowAI is integer rather than floating number. In conclusion, matrix factorization is commonly used in the models we choose and models blending is important for us to get better prediction.

References

[1] Nicolas Hug.*Surprise Website*,[website]. Retrieved from: http://surpriselib.com/

[2] maciejkula.*Spotlight*,[Github]. Retrieved from: https://github.com/maciejkula/spotlight

[3] coreylynch.*PyFM*,[Github]. Retrieved from: https://github.com/coreylynch/pyFM

[4] Yehuda Koren (August,2009). *The BellKor Solution to the Netflix Grand Prize*, [PDF file]. Retrieved from: https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

[5] James Bergstra, Yoshua Bengio(2012). *Random Search for Hyper-Parameter Optimization*, [PDF file]. Retrieved from: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[6] Steffen Rendle(2012). *Factorization Machines with libFM*, [PDF file]. Retrieved from: https://www.csie.ntu.edu.tw/~b97053/paper/Factorization%20Machines%20with%20libFM.pdf