ML Recommender System - Project 2

Lamyae Omari Alaoui , Nessreddine Loudiy
Department of Computer Science, EPFL, Switzerland

***Abstract*—The purpose of this project is to present a recommender system model that predicts user's rating for movies using machine learning algorithms. In this paper we will introduce and present in details the algorithms we used. We will discuss the results and select the algorithm that fits the best the data we have. The Root Mean Square Error (RMSE) score is used to evaluate the performance.**

I. Introduction

Providing useful suggestions of products to online users in order to increase their consumption on a given website is the goal of many companies nowadays such as Netflix, Alibaba, etc. The recommender systems are tools that provide suggestions that best suit the client's needs, even when they are not aware of it. The applications of recommender systems include recommending movies, music, television programs, books, etc.

The aim of this project is to predict user ratings for movies based on a dataset of some already known ratings and without having any additional information on movies nor on users. We will explore several methods and compare their performance to choose the most accurate one.

The input of our algorithm is the matrix X (a $D \times N$ Matrix), where the entry (d,n) is the rating of the movie d given by the user n, if the rating is known, zero otherwise. The metric we use to validate our model is RMSE defined as:

$$RMSE := \sqrt{\frac{1}{|\Omega|} \sum_{(d,n)\in\Omega} \frac{1}{2}(x_{dn} - \hat{x}_{dn})^2} \qquad (1)$$

Here Ω is the set of the indices of the known ratings of the input matrix X, x_{dn} is the true rating and $\hat{x}_{dn}$ is the predicted rating provided by the algorithm.

II. Dataset

We will use the movie dataset from kaggle, which contains 1176952 ratings of 1000 users for 10000 different movies which represent less than 15% of the full rating matrix. The ratings are integer values between 1 and 5. The goal is to find a model that can accurately predict the unseen ratings.

The following figures gives us an idea of how many ratings each user gave, and how many ratings each movie received. It's very useful to look at the minimum number of given/received ratings in our dataset. In our case, the minimum number of ratings given by a user is 8 ratings and each movie has received at least 3 ratings. Before using these data in our model, we're going to clean it and discard all the users and movies with less than 20 ratings. This preprocessing will allow us to use only meaning full data while tuning the parameters of our model.

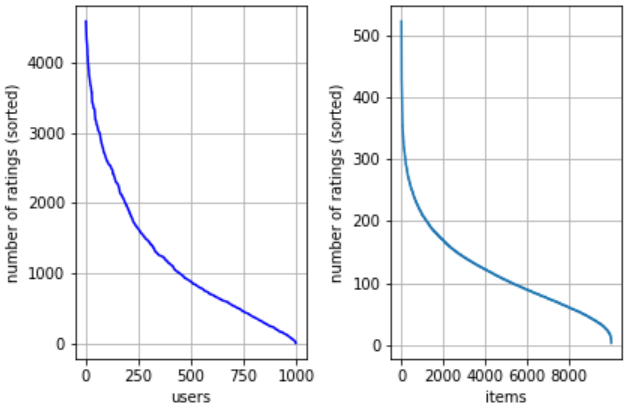

Figure 1. The left plot shows the number of ratings per user ,while the right one shows the number of ratings per movie.

III. Baselines

This part was done before implementing the other methods in order to have a reference to which we can compare our results.

- Global mean : we predict all the unseen ratings with the value of the global mean .The score on AICROWD was:1.127.
- User mean : we predict the value of a rating for a user n with the mean of his observed ratings i.e the mean of column n.the score was:1.07
- Item mean: we predict the value of a rating for a movie d with the mean of its ratings i.e the mean of row d.the score was:1.117

IV. Classification method

Our first approach is to use classification to solve this problem. We will try to classify each entry (d,n) into one of the classes {1, 2, 3, 4, 5} corresponding to the rating of the movie d by the user n.

We used the Optimized Gradient Boosting algorithm, to solve our classification problem. Precisely, we are using the XGBoost library in python, which is a decision-tree-based ensemble Machine Learning algorithm. The word ensemble is used to denote that the prediction is done by aggregating

predictions of several models. It begins by initializing the ensemble with a single model, whose predictions can be pretty naive, but by adding new models, the predictions tend to get more and more accurate.

The features in our case are the users and the movies IDs, and the outputs are the ratings.

A. Tuning

To improve our model, we need to tune its parameters. XGBoost gives us a variety of parameters we can configure to fit best our data. The parameters we chose to tune are:

- n_estimators = 1000: The number of models that we include in the ensemble.
- learning_rate = 0.2: It's a multiplication factor that we apply to each new model we add.
- early_stopping_rounds = 5: Early stopping causes the model to stop iterating when the validation score stops improving, even if we aren't at the hard stop for n_estimators. This is why we set n_estimators as high as 1000 cycles.

B. Results

Using this classification approach, we get a 1.066 RMSE score in the Aicrowd plaform. A very low score compared to the other methods we used. This can be explained by the very few number of features we have used to predict the rating, as we do not have access to any additional informations about users or movies besides their IDs. And since XGboost algorithm relies heavily on the features to classify each entry, we got a low score.

V. Matrix factorization

Another approach to solving this problem, is by using the matrix factorization, which is considered the state of the art method for solving recommender system problems since it's use in Netflix prize.

This method consists of decomposing the matrix $X \in \mathbf{R}^{D\times N}$ into the product of two rectangular matrices having dimensions of (D,K) and (N,K), where K represent the number of latent features. The aim of Matrix factorization is to find $W \in \mathbb{R}^{D\times K}$ and $Z \in \mathbb{R}^{N\times K}$ such that $X \simeq WZ^t$ (i.e finding W and Z that minimize the RMSE loss function). We refer to W as the item matrix and to Z as the user matrix. We define our cost function as follows:

$$\frac{1}{2}\sum_{(d,n)\in\Omega}(x_{dn}-(WZ^t)_{dn})^2+\frac{\lambda_w}{2}\mid\mid W\mid\mid+\frac{\lambda_z}{2}\mid\mid Z\mid\mid \tag{2}$$

We can not assume that the global mean is unique since the cost function is not convex. In order to solve this optimization problem, we will use three algorithms: Stochastic Gradient Descent (SGD) , Alternating Least Squares (ALS) and Neural Networks.

A. Stochastic Gradient Descent

The gradient descent is an optimization algorithm which is mainly used to find the minimum of a function. However, in our problem, the matrix is very large which makes this algorithm computationally expensive. So, we opt for a stochastic gradient descent algorithm.

1) Matrix factorization initialization: We start by initializing the user and the item matrices randomly. For the matrix Z we assign the mean of each movie as the first row. We do the same thing to the user matrix W, by assigning the first row to the mean rating of each user. By experience, we found that this initialization, made our predictions better, compared to a totally random initialization. We improved our score from 1.044 to 1.041 just by doing so.

2) SGD update algorithm: The derivative of the cost function (without the regularization term) can be written for each element of the matrices W and Z in the following way:
if d=d':

$$\frac{\partial}{\partial w'_d,k}f_d,n(W,Z)=-(x_{dn}-(WZ^t)_{dn})z_{n,k} \tag{3}$$

0 otherwise
and:
if n=n':

$$\frac{\partial}{\partial z'_n,k}f_d,n(W,Z)=-(x_{dn}-(WZ^t)_{dn})w_{d,k} \tag{4}$$

0 otherwise
At each iteration , the elements of W and Z are updated as follow:

$$W^{t+1}=W^t-\gamma\nabla_w f_{d,n} \tag{5}$$

$$Z^{t+1}=Z^t-\gamma\nabla_z f_{d,n} \tag{6}$$

The cost function without regularization did not perform very well, this might be due to its non convexity. In practice, the iterations are performed until reaching the max iterations that we defined. Finally , the predictions can be found by computing WZ^t. The algorithms seems to give good results but it is very depending on the parameters γ, λ_w ,λ_z , also the number of iteration and the number of features plays an important role.For this reason , we decided to do a grid search function in order to choose them wisely ,i.e we iterate over the different values of the parameters and at the end the function returns the combination of the parameters with the smallest RMSE.the γ parameter is divided by 1.2 at each iteration to help the algorithm converge to the minimum.

B. Alternating Least Squares

1) Matrix factorization initialization: We use the same initialization as in the SGD case.

2) ALS update algorithm: The idea behind the alternating least square algorithm, is to consider W fix when updating Z and vice versa. This way, we alternatively update W and Z. This method is of a great interest when working with huge datasets, as it can perform both updates in parallel, which saves us a great amount of time.

The update schemes of Z and W are:

$$Z = (WW^T + \lambda_z \times I)^{-1} \times WX \tag{7}$$

$$W = (ZZ^T + \lambda_w \times I)^{-1} \times ZX^T \tag{8}$$

Here we are not aiming a maximum number of iterations, but we're going to use a stopping criterion instead. We will stop when the error variation is smaller the a fixed number that we define.

C. Neural networks

In this section , we will use the factorization model of the python framework Spotlight.it also used the idea of decomposing the matrix of ratings into two matrices i.e user and item matrices .Spotlight uses Pytorch to build deep recommender models.This library creates objects that describe the user-item interactions ,user ids, item ids, and ratings should be provided for all observed ratings in the dataset.We decided to use the explicit feedback factorization model which use a classic matrix factorization approach as describes in the previous section by representing both users and items with latent vectors.To get the prediction , we use the dot product.For the parameters,we decided to use L2 ridge regression as a loss function and a number of latent features equal to 20.

VI. Surprise

In this part, we will use the surprise library built by Nicolas Hug .it enables us to analyse and built recommender system .We tried to benchmark the following algorithms :

- Normal predictor
- Beseline only
- k-NN algorithms i.e KNNBasic ,KNNWith means ,KNNWithZscore,KNNBaseline
- Matrix factorization algorithms ,i.e SVD ,SVD++ ,etc
- Co-clustering

VII. Results

In this section ,we will present our results for the different algorithms described before for the matrix facotrization approach.

A. SGD

As we said before ,SGD is very sensitive to the hyper parameters .From the first experiments , we found that a large number of feature gives best results.We decided to use the grid search function to find the best parameters :

- K =5
- γ= 0.009
- λ_w=0.24
- λ_z=0.01

With this combination , we acheives an RMSE=1.041 on AICROWD.

K	Epochs	γ	λ_w	λ_z	RMSE
5	10	0.009	0.24	0.01	1.0034
1	20	0.02	0.1	0.1	1.048
1	20	0.035	0.1	0.1	1.04855
5	100	0.005	0.015	0.02	1.0035

B. ALS

The result of our model can be improved by choosing the right parameters. We used a number of features K=50 and we will stop our algorithm whenever the error variation is smaller than 10^{-5}. Then we are going to tune the lambda user and the lambda item to have the best test RMSE we can get. Here is a summary of how the test RMSE varries when we tune these parameters:

K	Stopping criterion	λ_w	λ_z	RMSE
50	10^-5	0.1	0.095	1.04255
50	10^-5	0.1	0.1	1.04055
50	10^-5	0.1	0.15	1.03955
50	10^-5	0.1	0.2	1.05633
50	10^-5	0.01	0.15	1.129998
50	10^-5	0.08	0.15	1.03677
50	10^-5	0.1	0.15	1.036147
50	10^-5	0.2	0.15	1.04630
50	10^-5	0.3	0.15	1.06021

C. Neural Networks

We tried different parameters to test this algorithm, the table below shows the important results.

K	Epochs	γ	λ	Batch$_size$	RMSE
5	8	1e-5	1e-10	2000	1.00085
150	10	1e-4	1e-9	1500	1.004
20	30	1e-4	1e-5	32	0.92

With the parameters of the last line ,we achieved our best score of 1.032 on AICROWD.

D. surprise

We have used this library in order to analyse the results that different algorithms will give .The results are shown in the figure below: As we can see The method BaselineOnly

Algorithm	test_rmse	fit_time	test_time
BaselineOnly	0.999947	3.832720	6.362024
SlopeOne	1.001506	6.460053	60.607916
KNNBaseline	1.011970	120.580732	847.633581
CoClustering	1.013733	29.899863	6.541970
NMF	1.014375	77.733584	5.737661
KNNWithMeans	1.024882	111.466288	792.832491
SVDpp	1.024894	1674.849502	69.053940
KNNWithZScore	1.026487	115.130776	851.329665
KNNBasic	1.031897	100.597629	715.553122
SVD	1.032499	74.655963	7.463581
NormalPredictor	1.483146	2.329978	7.345178

Figure 2. The Benchmark using surprise

,which predicts the baseline estimate for given user and item, gives the best results.

VIII. Discussions

After tuning all our models, we can now compare the performance of each of our models on the given dataset.

The final results show that neural network outperformed all the other algorithms who's submission yielded a RMSE score of 1.032, followed by the ALS algorithm with 1.036, then SGD with 1.041 and finally the classification approach with 1.066.

The ALS and SGD methods depend strongly on the matrix factorization initialization, as we could go to the global minimum if we were lucky to find a good starting point. Performing the calculations in parallel made the ALS calculations faster than SGD ones.

The use of Apache spark could have helped us to distribute calculations over the different cores of the GPU to get a better calculation performance and to minimize time.

The classification approach would be more suitable in case we have many features for both the users and movies. As we only have two features, which are the IDs of users and movies, we couldn't cluster the users and movies perfectly and thus the prediction wasn't good enough.

Another approach that we haven't tried, is combining multiple methods instead of just using each one independently.

IX. Conclusion

We managed to achieve a result of 1.032 using our neural network in the AIcrowd platform. This model that performed better than the other models. Even though ALS and SGD both minimize the same loss function, alternating updates between W and Z lead to a better score. The classification approach was not able to compete with the other models for this particular case, which could be explained by the lack of additional features and informations about the users and movies.

Better results could be achieved by combining their ideas into a single model.